1990

Design and implementation of a multimedia DBMS : sound management integration

Atila, Yavuz Vural

Monterey, California: Naval Postgraduate School

http://hdl.handle.net/10945/27548

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND IMPLEMENTATION
OF A MULTIMEDIA DBMS :
SOUND MANAGEMENT INTEGRATION

by

Yavuz Vural Atila

December 1990

Thesis Advisor: Vincent Y. Lum

Approved for public release; distribution is unlimited.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School	6b. OFFICE SYMBOL (if applicable) 37
7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (if applicable)
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Design and Implementation of a Multimedia DBMS: Sound Management Integration

12. PERSONAL AUTHOR(S)
Atila, Yavuz Vural

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 12/89 TO 12/90	December, 1990	113

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Multimedia Database Management System, Multimedia, DBMS, MDBMS, Sound Media Management, Sound Database

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
Today, in addition to standard data like numerics and alphanumerics, it is possible to capture , store, manage, retrieve and present different media information, e.g. sound, image, graphics, text and signals, by using the current, modern computer technology. The Multimedia Database Management System (MDBMS) project, which started at the Computer Science Department of the Naval Postgraduate School in 1988, studies not only the storing, managing and retrieving different media information, but also the management of the interrelationships among the data. This thesis concentrates on the Sound Management Integration of the MDBMS prototype, which includes the storage and management of the sound records in an IBM compatible personal computer, and the connection and integration of it to the database system in the SUN environment through a local area network, after presenting the general overview of the MDBMS, the system environment, the user interface and the catalog designs.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Prof. Vincent Y. Lum	22b. TELEPHONE (Include Area Code) (408) 646-3091 — 22c. OFFICE SYMBOL CsLm

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited.

DESIGN AND IMPLEMENTATION OF A MULTIMEDIA DBMS:
SOUND MANAGEMENT INTEGRATION

by

Yavuz Vural Atila
Lieutenant JG, Turkish Navy
B.S., Turkish Naval Academy, 1984

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1990

Author: [redacted]

Yavuz V. Atila

Approved by: [redacted]

Vincent Y. Lum, Thesis Advisor

[redacted]

David Hsiao, Second Reader

[redacted]

Robert B. McGhee, Chairman
Department of Computer Science

ABSTRACT

Today, in addition to standard data like numerics and alphanumerics, it is possible to capture, store, manage, retrieve and present different media information, e.g. sound, image, graphics, text and signals, by using the current, modern computer technology. The Multimedia Database Management System (***MDBMS***) project, which started at the Computer Science Department of the Naval Postgraduate School in 1988, studies not only the storing, managing and retrieving different media information, but also the management of the interrelationships among the data. This thesis concentrates on the Sound Management Integration of the MDBMS prototype, which includes the storage and management of the sound records in an IBM compatible personal computer, and the connection and integration of it to the database system in the SUN environment through a local area network, after presenting the general overview of the MDBMS, the system environment, the user interface and the catalog designs.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I am grateful to Dr. Klaus Meyer-Wegener, Dr. C. Thomas Wu, Gregory R. Sawyer and Cathy A. Thomas, who have made great contributions to the development of the MDBMS system and have thus provided me the opportunity to work on this important database project. I would also like to thank Dr. Kyung-Chang Kim and my thesis companions Su-Cheng Pei and Wuttipong Pongswuan, with whom I worked together as a team to design the current MDBMS prototype.

I would like to express my sincere thanks to Dr. Vincent Y. Lum for all his support and encouragement in the conception and preparation of this thesis, and his inexhaustible efforts in the development of the MDBMS project.

I. INTRODUCTION

A. BACKGROUND

Today, in addition to standard data like numerics and alphanumerics, it is possible to capture, store, manage, retrieve and present different media information, e.g. sound, image, graphics, text and signals, by using the current, modern computer technology. Multimedia Database Management System (***MDBMS***), beyond storing, managing and retrieving different media information, also manages the interrelationships among the data. MDBMSs have found its way to many application areas like classroom teaching, military command/control and training, libraries, archives, communication, publishing, advertising, and computer-integrated manufacturing [Lo88]. These application areas are nonstandard for the conventional DBMSs, because DBMSs have been developed to manage only the standard, structured alphanumeric data. On the other hand, MDBMS requires both the management of alphanumeric and multimedia data together.

Multimedia or media data have been introduced as text, graphics, images, sound, and signals. They all have in common in that a single "value" or object of that type tends to be rather *long*, i.e. in the range of 100K to 10M bytes. They are often referred to as *unformatted* (unstructured) data, meaning that they consist of a large and varying number of small items, like characters, pixels, lines or frequency indicators stored together in some way to form a unit. They all carry a more complex structure which varies significantly from value to value and is often not known to the user when the object is

stored. Detecting the type of media, which has been stored, requires some level of understanding and recognition. More detail information about the characteristics and storage requirement of media data can be found in [MLW88]. Due to high storage requirement of media data, we are only able to keep them in separate files with the current available technology. They can be a separate text, a separate image or sound, which we call a "media object" throughout this thesis, but actually each of them is only a value of an instance in multimedia data. In multimedia data, a sound for example is an object, but it is also the value of the attribute voice.

The principal task of the DBMSs is storage and retrieval, but not processing the data. From this point of view, the big effort concentrates on the storage and retrieval of the multimedia data by the content of the data, which is processed. Handling *content search* in multimedia data is a difficult problem; one needs to find ways to handle a very large amount of multimedia data and to search and find the appropriate data, conveniently and efficiently based on the contents of the media. Without the ability to do content search, having a MDBMS would be like having a DBMS that cannot process queries on standard data. The content search issue on media data is not possible with the current methods used on formatted data structure, so we need to use an *abstract data type* concept [LM89]. Image, sound, signal, text and graphic data will be treated as new data types. Any attribute of an object can have one of these data types, instead of the usual data types like characters, integer, boolean etc. We also need to define operations to process (create, retrieve, update, delete) these new media data "values". The MDBMS Project in the Computer Science Department of the Naval Postgraduate School was

formed to develop a technology that would allow us to handle multimedia data as conveniently as we can process the standard data, with the emphasis of providing content search capability [WK87, LM89].

Besides the MDBMS Project at Naval Postgraduate School, there are a number of researches going on in multimedia data processing around the world. Among those, the *MINOS* project at University of Waterloo is an object-oriented (*O-O*) multimedia information system that provides integrated facilities for creating and managing complex multimedia objects [Ch86]; an O-O database management system named *ORION* has been developed at MCC in Austin/Texas, which contains a Multimedia Information Manager (MIM) for processing multimedia data [WK87]; the IBM Tokyo Research Laboratory has developed two "mixed-object database systems", which are named as *MODES1* and *MODES2* [KKS87]; and in Europe there is an *ESPRIT* project designing a multimedia filing system called *MULTOS* [Be86, BRG88]. A discussion of these projects is presented in [MLW88, LM89] and will not be repeated here.

Recently multimedia management in the personal computers becomes available by using *hypertext* and *hypermedia*. The concept of hypertext is very old; it has been transferred to computer systems since 1960's. Originally intended to manage arbitrarily linked text segments, it has been extended to manage images and sound, and has become "Hypermedia" [MLW88]. The hypertext and hypermedia data management in the Macintosh computer with a *hypercard* application has many users, including the *ARGOS* project being developed at Naval Postgraduate School [WNTA89]. The hypertext and hypermedia data management uses the hierarchical data structure approach, in which users

cannot query the data as done in the conventional DBMSs, but have to follow the hierarchical tree structure to process a media. As a result, the users might easily get lost during a process. Additionally, hypertext requires an interpreter to process the user commands. Furthermore, the hypertext and hypermedia data cannot be accessed by the other users, as in the database systems, because they are designed to work only on personal computers in the single user environment. MDBMS, which is a DBMS introduced in [LM89] with the extended capability to process the multimedia data, was designed to overcome the restrictions and disadvantages of hypertext and hypermedia systems.

B. AN OVERVIEW

MDBMS mainly consists of two subsystems: one to manage the structured data in the conventional database environment (*INGRES*), and the second one to process the multimedia data in an extended DBMS environment. These two subsystems are integrated to provide a uniform interface to process structured or multimedia data, or both together.

The general design of the overall application for MDBMS includes the main, high level database operations like *table creation*, *data insertion*, *retrieval*, *deletion* and *modification*. Three students are doing related work on their M.S. theses in the Computer Science Department of the Naval Postgraduate School. The detail design and implementation for table creation and tuple insertion along with catalog management, are presented in the thesis by Pei [Pe90]. The design and implementation for retrieval operation can be found in the thesis by Pongsuwan [Po90]. In this thesis, we will

concentrate on the storage and management of *sound data* using an IBM compatible personal computer, connected to the main database system through a local area network.

In the following chapters, first we will concentrate on the MDBMS system and then talk in detail about sound management integration. In Chapter II, we will discuss the following subjects: the development of the MDBMS project before this thesis, including the architecture of the system; the system environment in which the MDBMS is built; and the software and hardware requirements. In Chapter III, we will discuss the high level user interface design; the MDBMS catalog design; the processing of structured data in INGRES and the processing of multimedia data in the extended system, which currently includes sound and image; and the parser that is used to process natural language descriptions of the media data. Chapter IV concentrates on an overview of sound characteristics and the sound management in the PC, the details of hardware and software requirements for sound management, and the sound management user interface design. In Chapter V, we will talk about the implementation details of sound management in both the IBM compatible PC and the SUN environments, and the PC-SUN linkage. Chapter VI will summarize and present the conclusions.

The program codes for the sound management in PC and the codes for the sound management integration to main MDBMS program are included in the Appendices A and B respectively.

II. SURVEY OF THE MDBMS

A. DEVELOPMENT OF MDBMS

The work in the MDBMS Project at the Computer Science Department of the Naval Postgraduate School started with the design of the architecture of the MDBMS in 1988 [LM88, LM89]. The main goal was to have a database system that can process the multimedia data as conveniently as the processing of the standard (structured) data. The architecture of the system is composed of the three main parts: the *MDBMS User Interface* which is the interface between the MDBMS user and the formatted and media data managers, the *Standard DBMS* which manages all the queries related to the formatted data, and the *Multimedia Manager*, which deals with the multimedia data. One may consider that the Multimedia Manager to be composed of different subsystems related to the various media data types as image, sound, text, graphics and signals. The current MDBMS includes only the *Image Manager* and *Sound Manager*, since only image and sound data are supported at this time. The architecture of the system is shown in Figure 1, and the detail discussion about the MDBMS architecture is presented in [LM88]. Media data management requires the processing of both the structured and unstructured data together, because it is not possible to handle media data by itself. For multimedia data processing, the *abstract data type (ADT)* concept has been determined as the most appropriate model for the task [LM88]. It was proposed that media data types like image, sound, text, graphics and signal be defined and their processing operations constructed.

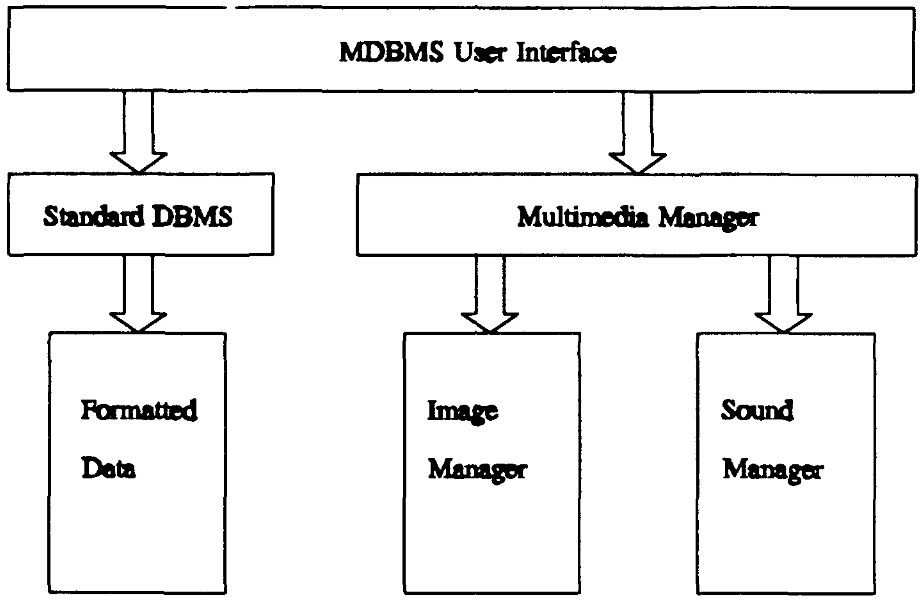

Figure 1. The Architecture of the MDBMS

The new system will be able to support the operations through this structure, processing the formatted and multimedia data. For example, the system can now process data in the relation EMPLOYEE(name, age, salary, photo) where name, age and salary are the formatted data types, and photo is the image media data type. The detail operations on the image data type is given in [LM88].

Raw media data like a pixel matrix of an image, or the bit string representation of a sound or signal are obtained from the process of digitizing the original media object. Processing the raw data needs information like resolution, pixel depth, colormap, sampling rate, etc. that describe the techniques used to capture/encode the media data. This textual data part was named as *"Registration Data"*. In addition to registration data, another textual description, named as *"Description Data"*, has been attached to the raw data for

the purpose of performing content search, because automatic recognition of the media data by computers is beyond the state of the art today. [LM89]

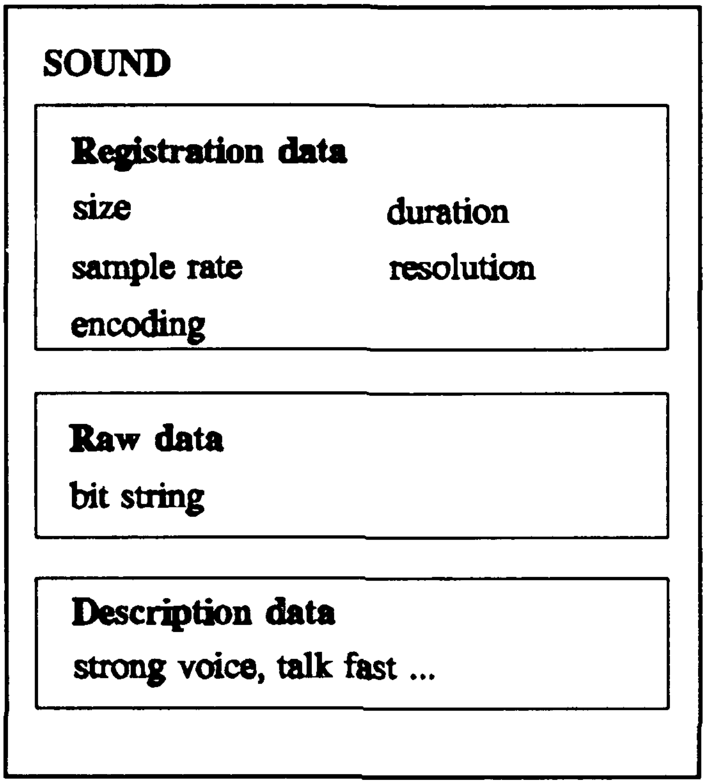

Figure 2. Structure of a Media Object (SOUND)

This media object model essentially consists of three parts: registration data, raw data and description data, as shown in Figure 2. The registration data is necessary to interpret or display, identify or distinguish a definite raw data from others. The registration data may have a fixed format, because the formats or field lengths of information required to access the different media data are known.The description data, that describes the object representation of the raw data and is used for content search purposes, generally can not be derived by computers and must to be entered by the user in *natural language form*. To understand these descriptions, the system requires a fairly

rich and sophisticated dictionary, depending on the application. An application naturally limits the scope of this dictionary size since vocabulary not used in the application should not be entered.

The image and sound media data types are already defined [Th88,Sa88]. Both have their own sets of operators or functions for processing the registration data, the raw data and the description data. The registration data in the image ADT include file size, resolution, encoding, and colormap, and the registration data in the sound ADT include file size, sample rate, encoding, duration and resolution.

Processing description data requires sophisticated techniques of natural language processing.A *parser* was thus constructed for this purpose. Using the dictionary or lexicon prepared for the application transforms the description data in natural language form into a more formal representation, namely a set of predicates and literals, suitable for *Prolog* processing. These predicates state a fact about the content in the media object. Thus, the set of all predicates that can be used in the descriptions must be defined in the dictionary. When a query is entered for media data retrieval, it must be also in natural language form and be parsed by the parser. A media object is selected as the result of the query, if and only if the media object description logically implies the query description. The detail information about the parser is presented in [Du90].

The implementation of the MDBMS began in 1988. The subcomponent Image Manager of the Multimedia Manager was initiated by a M.S. thesis student Thomas [Th88]. Thomas provided the low level functions for processing image database like storing images in a relational DBMS, retrieving registration and raw data for display, etc.

Subsequently another student [Po90] implemented the high level retrieval operations using the low level fun. .ons defined by Thomas [Po90].

The subcomponent Sound Manager of the Multimedia Manager, as shown in Figure 1, was implemented by another M.S. thesis student Sawyer [Sa88]. Sawyer provided a similar processing capability for the incorporation of sound media data type as done by Thomas for the image data. He used an IBM compatible PC for processing the sound data, because SUN 3 Workstation did not have the sound processing capability at that time.

The content search for the media data was first implemented by Meyer-Wegener [LM90] by using a parser implemented by another faculty in the department to parse the natural language descriptions entered by user and get the result in an acceptable form by Prolog. The initial parser was relatively simple and could parse only phrases. Subsequently the parser was greatly modified to have the capability to parse complex sentences and structures [Du90].

The first implemented MDBMS prototype constructed on top of the INGRES DBMS was designed for managing only the image media data types in the SUN 3 Workstation environment, and did not have the capability to process both formatted data and multimedia data. It could retrieve the image media by using the identifiers of the image data, or their natural language descriptions.

The current MDBMS prototype is to broaden the database handling capability of the system, by providing the integrated support for both formatted data and multimedia data. Its design and implementation is based on the same architecture of the previous

work, and it will provide the high level operations of table creation and data insertion, retrieval, deletion, and update for both formatted and multimedia data. Three thesis students worked on these high level operations. Pei [Pe90] worked on the table creation and data insertion, Pongswuan [Po90] worked on retrieval process, and in this thesis we will talk about the *Sound Management Integration* of this prototype. The detail explanations about the design and implementation of this new prototype will be presented in the following sections of this chapter.

B. SYSTEM ENVIRONMENT

Due to various resource constraints the decision in 1988 was to construct a MDBMS on an existing database management system. INGRES DBMS was chosen for this purpose and the SUN workstations and servers with the UNIX operating system were chosen to be the system to construct the MDBMS in. Because the SUN workstations did not support the sound management at that time, an IBM compatible PC was used to store and manage sound data. A parser was built for the content search purpose. The data connection between the SUN workstation and the other stations like UNIX, servers, parser and IBM PC is established by using *Ethernet*, a local area network. The general layout of the system is shown in Figure 3.

A number of restrictions are the consequence of using INGRES. First, the INGRES version in which the original MDBMS prototype was constructed does not support user-defined abstract data types. Second, INGRES allows maximum 500 characters to be stored for a given attribute. Third, it does not allow its user to get the catalog information

readily. Fourth, an intermediate interface below the SQL language is not available in INGRES. The restrictions mentioned above affected the design and implementation of MDBMS. [Po90]

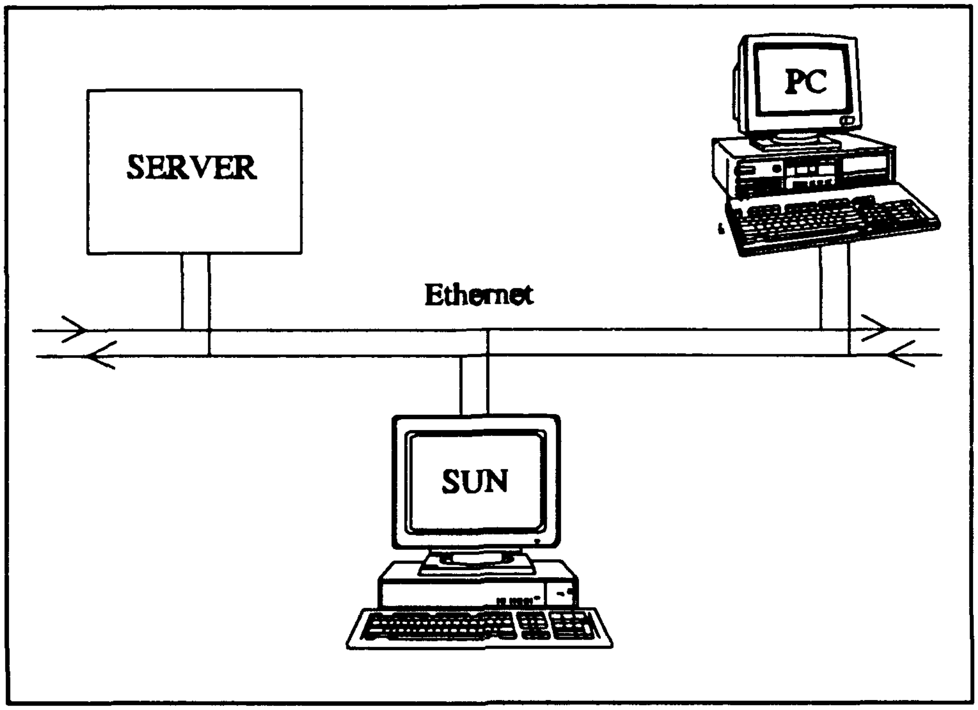

Figure 3. The MDBMS System

In the meantime, the capabilities of INGRES and SUN workstations have changed. Now, some of the INGRES restrictions mentioned above have been removed, and SUN can support sound. As the MDBMS prototype is not intended to be a production system at this time, a decision was made not to change the structure of the system, because the current system is enough for the demonstration of the various concepts. Otherwise it would require substantial investment to purchase new hardware and recode some written software. It was decided that, instead of these investments, which would provide little gain, the IBM compatible PC system would be retained to manage sound data and would

be integrated into the MDBMS prototype in SUN environment as a backend server for sound management via a local area network connection, which in this case is the *Ethernet*.

For image capturing process, a video card which works with a camera recorder was installed in a PC. After capturing the images, the image files are transferred into a SUN workstation and processed.

The system environment just discussed influences the design and implementation of the system and will be reflected in the various parts in this thesis, as well as the theses by Pei [Pe90] and Pongsuwan [Po90]. However, the complexities existing in the system's configuration are transparent to the users. They will find all the operations as if they all belong to only one system.

III. INTERFACE AND CATALOG DESIGN OF THE MDBMS

The MDBMS prototype user interface design that includes the high level DBMS operations and applications, and catalog design that includes the catalog tables and their interrelationships, will be discussed in this chapter. As previously mentioned in the introduction chapter, two more M.S. thesis students have been involved in the current design of the MDBMS, and consequently the subjects in this chapter are also presented in varying depth and breadth in the companion theses [Pe90, Po90].

A. USER INTERFACE DESIGN

1. The Main Operations

In this section, we will discuss the MDBMS from the user's point of view. The allowed operations in the MDBMS are the same high level operations as in a standard DBMS for formatted data, namely table creation, data insertion, retrieval, modification and deletion. Currently, most the first three operations have been implemented, and the rest of them are in progress [Ay91, Pe91, St91].

Because the MDBMS is composed of two different database managers, which are INGRES for formatted data and the multimedia manager for media data, an extended SQL language structure is needed to specify the queries [LM89]. But instead of asking a user to enter queries in formal SQL structure, the user interface is designed to get these required data interactively in a user friendly way. After getting the required data for

queries, the MDBMS system itself creates the formal SQL statements and executes the requested operations.

After running the MDBMS program, which is written in "C" language, the user gets the main menu as shown in Figure 4. First we will try to explain each option in numerical order and then show the application of interface with a couple of examples in the next section.

Multimedia Database Management System

1. Create Table
2. Insert Tuple
3. Retrieve
4. Delete
5. Modify
0. Quit

Figure 4. The main menu of MDBMS

The first option is the *create table* operation, which is used to create relational tables in the MDBMS. It allows a user to enter the table name, and then the attributes which have data types like character, integer, float, and media types image and sound. The program displays a warning message in case of table name duplication. After inserting all the data about the new table, attributes, and their data types, the user has an

opportunity to modify the entered data, and insert more attributes or delete already inserted attributes.

The *insert tuple* operation allows the user to enter tuples one at a time. When the user enters the table name to insert tuple, the process starts and the system asks the user to enter interactively the values for the attributes in the given table, in the sequence according to the order defined in the relational table. For media data, which should be transferred to the MDBMS environment before insertion, the insertion process is different than that for the formatted data. The program needs the media data file name, which includes the registration data and the raw data, to fetch the registration and raw data and to put this information into the media data relation. As explained in [Pe90], the media relation is created by the system and is hidden from the user. To complete the third part of the media object, the description data, the user has a chance to see the image or to listen the sound record before inserting the description in natural language form. Again, after the completion of all the attributes the system displays the inserted attributes values to get a confirmation from the user, and gives the user a chance to change the values, if so desired.

The *retrieve* operation is used to get the required attribute values of the tuples with respect to the user defined conditions, which coincide with the WHERE clause in SQL statements. First, the user enters the table names, which relate to his/her query. If the users has more than one relational tables, then he/she has to enter join conditions to connect the tables. Next, the user enters the attribute names for each relational table. The system checks each attribute names entered for their existence in the given table, and

displays a warning message in case of a non-existing attribute. The next step is to enter the conditions to retrieve the desired information from the related tables. The current MDBMS prototype supports the boolean conditions in *disjunctive* normal form. It uses the **AND** boolean operator inside each group, and the **OR** boolean operator between group conditions. A group condition is either a single condition or multiple conditions grouped together. Conditions not specified in the disjunctive normal form are not supported in the current MDBMS prototype. The reason for preferring the disjunctive normal form is to simplify implementation without sacrificing functionality and usability. The condition statement for the formatted data is entered with a preceding operator like <, >, =, <>, >= etc. and additionally, for the character data types the text data should be in quotation marks following the equal sign operator like (="......"). For the media attributes, the media description must be entered without quotation marks. Multiple descriptions must be entered one description at a time followed by a blank line, when no more description is to be entered. After these interactive inputs, the system processes the queries and displays the requested data in a table form. However, while the values for the formatted attributes in the display are really the desired values, the values for the attributes of media data types are only index numbers which points to the tuples in the media relations, and cannot be understood by the user. They are created and used by the system itself. In case of the existence of media data, the system asks to the user whether to display the media data or not, in a sequential order.

The *delete* operation is used for deleting the specific tuples in a table by entering condition statements, or deleting all tuples and afterwards dropping the relation without

any condition. Constructing the condition statements is the sa^Xx as explained above for the retrieve operation.

The last one, the *modify* operation, allows the user to perform various operations: to update tables and attribute names like changing the table names or the attribute names in the table; to insert new attributes to the table; to update the formatted data inserted previously by entering the conditions and then the attribute names, the values of which the user wants to change; and to update the media data by entering the media file name after specifying the conditions and media attribute name. After changing the media file name, the user has an opportunity to update the description of the media data as well.

After the completion of each high level MDBMS operation, the system displays the main menu on the screen. The user can select either another operation or the *quit* option to exit the system.

2. MDBMS Applica.ions

Now, we will go through the user interface design of the MDBMS, by using the create table, insert tuple and retrieve operations, which are already implemented. For example, we want to create a relational table **SHIP**, with the attributes S_NAME, TYPE, S_NO, YR_BUILT, DISPLACEMENT, CAPT_ID, PICTURE. First, we run the MDBMS system and get the main menu on the screen and select the create table option, and then we will interactively enter the data in the following order (the *italics* represent the user's responses):

Enter table_name : (Maximum 12 characters)

SHIP *<cr>*

Enter attribute name : (Maximum 12 characters)

S_NAME <*cr*>

Select data type of attribute :

Select :: (1)integer (2)float (3) c20 (4)image (5)sound

Select your choice :: *3* <*cr*>

Data type : c20 (20 characters) (y/n):: *y* <*cr*>

More attribute in the table? (y/n) :: *y* <*cr*>

Enter attribute name : (Maximum 12 characters)

TYPE <*cr*>

Select data type of attribute :

Select :: (1)integer (2)float (3) c20 (4)image (5)sound

Select your choice :: *3* <*cr*>

Data type : c20 (20 characters) (y/n):: *y* <*cr*>

More attribute in the table? (y/n) :: *y* <*cr*>

....

Enter attribute name : (Maximum 12 characters)

CAPT_ID <*cr*>

Select data type of attribute :

Select :: (1)integer (2)float (3) c20 (4)image (5)sound

Select your choice :: *2* <*cr*>

Data type : integer (y/n):: *y* <*cr*>

More attribute in the table? (y/n) :: *y* <*cr*>

Enter attribute name : (Maximum 12 characters)

PICTURE *<cr>*

Select data type of attribute :

Select :: (1)integer (2)float (3) c20 (4)image (5)sound

Select your choice :: *4* *<cr>*

Data type : image (y/n):: *y* *<cr>*

More attribute in the table? (y/n) :: *n* *<cr>*

Table Name :: SHIP

Order	Attribute	Data Type
1	S_NAME	c20
2	TYPE	c20
3	S_NO	c20
4	YR_BUILT	integer
5	DISPLACEMENT	integer
6	CAPT_ID	integer
7	PICTURE	image

Any change before create? (y/n) :: *n* *<cr>*

Although the example above has no changes, the user could change table name, attribute names, and data types, or insert a new attribute into the table or delete an attribute from the table, if the user enters "y".

After creating the relational table, now we can insert tuples by choosing the second option, insert tuple :

Enter table_name : (Maximum 12 characters) (? for HELP!)

SHIP *<cr>*

Table Name :: SHIP
Att Name :: S_NAME

Data Type :: c20
Please Enter <<c20>> Value (? if unknown) ::

Kitty Hawk *<cr>*

Table Name :: SHIP
Att Name :: TYPE
Data Type :: c20
Please Enter <<c20>> Value (? if unknown) ::

Carrier *<cr>*

Table Name :: SHIP
Att Name :: S_NO
Data Type :: c20
Please Enter <<c20>> Value (? if unknown) ::

CV63 *<cr>*

....

Table Name :: SHIP
Att Name :: PICTURE
Data Type :: image
Please Enter <<image>> File Name !!
NOTE : Enter the Full Path :: (? if unknown)

/n/virgo/mdbms/.../....ras *<cr>*

Display the image before enter the description? (y/n) :: *y* *<cr>*

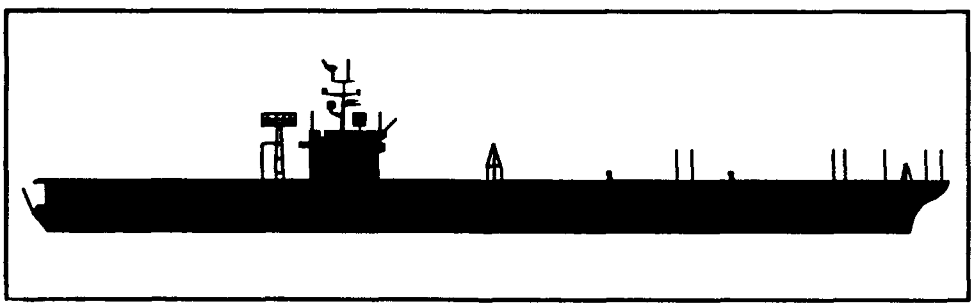

Figure 5. The image of USS Kitty Hawk

Enter the description? (y/n) :: *y* *<cr>*

Please enter the description :

NOTE: One phrase per line, end with an empty line ::

big aircraft_carrier with many airplanes
has many missiles of different kinds <cr> <cr>

Table Name :: SHIP

Order	Attribute	Data Type	Value
1	S_NAME	c20	Kitty Hawk
2	TYPE	c20	Carrier
3	S_NO	c20	CV63
4	YR_BUILT	integer	1961
5	DISPLACEMENT	integer	81123
6	CAPT_ID	integer	100
7	PICTURE	image	HAS VALUE

Media Description ::
Att_name :: Ship.picture
File_name :: \'90266.221112\'
Description ::
<<
big aircraft_carrier with many airplanes
has many missiles of different kinds
>>

Any change before insert? (y/n) :: *n*

Before inserting a tuple, the user has an opportunity to change the values for each standard attribute, and change the media file name or description. The process for the sound media type is the same. The system asks the user for the sound file name, whether to "Play the sound before enter the description? (y/n) :: ", and then to enter the description. The create table and insert tuple operations are presented in more detail in [Pe90].

The last MDBMS operation, that we will explain in the user interface environment, is retrieve. Let's assume that the user wants to retrieve the ships, built after 1960, with their names, types, numbers and pictures. For this retrieval, the query in extended SQL is:

```
SELECT s_name, type, s_no, picture
FROM   SHIP
WHERE  yr_built > 1960;
```

In the MDBMS system, the user enters this query interactively, after selecting the third operation from the main menu, which is retrieve. Upon completion of entering the query, the system creates the SQL statements and processes the query. An example of a session for entering a query is shown below:

Select the table(s), separate by comma <,> : *ship* *<cr>*

Table Ship

Select the attribute(s), separate by comma <,> : *s_name, type, s_no, picture* *<cr>*

4

ship.s_name
s_name c20

ship.type
type c20

ship.s_no
s_no c20

ship.picture
picture image

Any condition ? (y/n) : *y* *<cr>*

Group condition ? (y/n) : *n* *<cr>*

Enter attribute name : *yr_built* *<cr>*

ship yr_built
yr_built integer
Enter the condition

> *1960*

Select ship.s_name, ship.type, ship.s_no, ship.picture
From ship
Where yr_built > 1960

Record id 1 s_name: Kitty Hawk type: Carrier s_no: CV63 photo 1

Press ENTER to continue... *<cr>*

Record no 1 Filename: /n/virgo/work/mdbms/.../90266.221112

Number : 1
Description :
<<
big aircraft_carrier with many airplanes
has many missiles of different kinds
>>

Do you want to see the photo? (y/n) :: *n*

In the case the result contains more than one tuples when a query has been processed, the system asks the user whether to display media data like photo or sound one by one, in the order appeared in the resulting table.

Sometimes, to answer a query, the system must process more than one relational table, using *join condition* [EN89]. Let us illustrate this by assuming that, in addition to the SHIP relation; we have created a relational table **OFFICER** with the attributes O_ID, O_NAME, RANK, SALARY, PHOTO and VOICE. We will use CAPT_ID attribute in SHIP table and O_ID attribute in OFFICER table for the join condition. Now, let's

assume that the user wants to retrieve the ship names, officer names, ranks, photos and voices, for the officers whose rank is Lt Cdr and has soft voice (in description of sound). For this retrieval, the query in extended SQL will look as follows:

```
SELECT  SHIP.s_name, OFFICER.o_name, OFFICER.rank,
         OFFICER.photo, OFFICER.voice
FROM    SHIP, OFFICER
WHERE   (OFFICER.rank = "Lt Cdr" and CONTAIN (OFFICER.voice,
         "soft voice")) and (SHIP.capt_id=OFFICER.o_id);
```

Again as before, in this system the user enters this query interactively, after selecting the third operation from the main menu. The session will appear as follows:

Select the table(s), separate by comma <,> : *ship, officer* *<cr>*

Please enter the join condition : *ship.capt_id=officer.o_id* *<cr>*

Table Ship

Select the attribute(s), separate by comma <,> : *s_name* *<cr>*

1

Table Officer

Select the attribute(s), separate by comma <,> : *o_name, rank, photo, voice* *<cr>*

4

ship.s_name
s_name c20

officer.o_name
o_name c20

officer.rank
rank c20

officer.photo
photo image

officer.voice
voice sound

Any condition ? (y/n) : *y* *<cr>*

Group condition ? (y/n) : *y* *<cr>*

Enter the table name : *officer* *<cr>*

Enter attribute name : *rank* *<cr>*

officer rank
rank c20
Enter the condition

= *"Lt Cdr"*

End group (y/n) n <cr>

Enter the table name : *officer* *<cr>*

Enter attribute name : *voice* *<cr>*

Enter the sound description

"soft voice" *<cr>*

End group (y/n) *y* *<cr>*

More condition *(y/n)* *n* *<cr>*

At this point, the system converts the information given by the user into multiple SQL statements each of which is processed individually and the results coordinated and integrated to form the answer to the query. Again the users are asked whether or not displaying the photos and playing the sound records is desired. The detail information about this retrieval operation can be found in [Po90].

B. CATALOG DESIGN

The design and management of the MDBMS catalog are different from the INGRES catalog management. Since INGRES does not recognize the existence of media data, we have to design and build the MDBMS catalog ourselves,and manage the catalog as tables outside the INGRES. The decision was made to create the catalog in the form of system tables in the internal memory throughout the operation of MDBMS. When the user runs the MDBMS system, the files which contain MDBMS catalog tables are read into memory before any user operation is performed, and after the session the updated system tables are written out as files. [Pe90]

Table_Array

table_name	table_key	att_count	att_entry
SHIP	1	7	1
OFFICER	2	6	8
...	...	...	...

Table_List

1
2
...
...

Figure 6. MDBMS Table_Array and Table_List catalog tables

The MDBMS catalog is composed of three main tables or arrays: *Table_Array*, *Table_List* and *Att_Array* tables, as shown in Figures 6 and 7. The Table_Array contains the attributes: *table_name*, *table_key*, *att_count* and *att_entry*, where table_name denotes each different relational table, table_key denotes the number that will be assigned to the media in the create table operation, att_count denotes the number of attributes in the relational table, and att_entry gives the starting point of the table in the Att_Array table.

The second one, the Table_List array, contains the integer numbers which point to the entries in Table_Array, and it is created for database maintenance purposes. The last table, the Att_Array is composed of *att_name*, *data_type*, *media_id*, *next_index*, and *value_entry* attributes, where att_name denotes the name of the attributes in the created relations, data_type denotes the data type for each attribute including the formatted and media data, media_id column is set to "1" for media data types and "-1" for non-media attributes, next_index is the pointer to the next attribute in a given relation, and value_entry is the pointer to the Value_Array arrays.

Att_Array

index	att_name	data_type	media_id	next_index	value_entry
1	[illegible]	c20	-1	2	
2	type	c20	-1	3	
3	s_no	c20	-1	4	
4	yr_built	integer	-1	5	
5	displacement	integer	-1	6	
6	capt_id	integer	-1	7	
7	picture	image	1	-1	
8	o_id	c20	-1	9	
9	o_name	c20	-1	10	
10	rank	c20	-1	11	
11	salary	integer	-1	12	
12	photo	image	1	13	
13	voice	sound	1	-1	
...	...	...	...	...	

Figure 7. MDBMS Att_Array catalog table

The Value_Array arrays consist of five arrays and contain the attribute values for each attribute entry in the Att_Array table (Figure 8). There are three arrays for character, integer, and float data types, and two record arrays for media data (image and sound) representing the data types that the MDBMS supports. The value_Array tables are used to store the data before it is inserted to the database and also before it is displayed to the user.

C_Value	I_Value	F_Value
(char)	(int)	(float)

Img_Record

i_id	file_id	description	width	height	depth

Snd_Record

s_id	file_id	description	size	samp_rate	encoding	duration	resolution

Figure 8. MDBMS Value_Array tables

Now, for illustrating the use of the catalog tables, let us use the two relations named SHIP and OFFICER, which we have used for the user interface application in the previous section. First, when a new relation is to be entered, the catalog management part of the MDBMS will search through the relation names in Table_Array table, checking if

a duplication exists for the new relation name with the previous relations. In the case of any table name duplication, the user is given an opportunity to insert a new relation name. If no duplication exists, the catalog manager will put the relation into the next available row in the Table_Array table and transfer the row number of the new relation to the Table_List array. For instance, when we create the new relation OFFICER after the relation named SHIP, the system assigns the second row for our new relation, after checking for table name duplication. After that, the row (index) number, in this case 2, is transferred to the next available slot in the Table_List array, as shown in Figure 6. The following step is to enter the attributes and their data types of the relation OFFICER. The attributes and data types are entered into the Att_Array table, starting from the first available row in this table, and the corresponding index number of the first attribute is stored as att_entry attribute in the Table_Array table. For our example, the first attribute *o_id* and its data type are inserted to the eighth row in the Att_Array table (Figure 7), and the index number eight is transferred to the Table_Array table as att_entry (Figure 6). The remaining five attributes and data types are inserted to the Att_Array table one by one. The next att_entry value for a third relation is 14.

In the Att_Array table, the *next_index* attribute gives the index number of the next row. For implementation purposes, the next_index value of the last attribute of each created relation is an *end mark* (-1), instead of next row number. After the completion of the relational table creation, the user can modify the relation by changing the table name, attribute names and data types, or insert a new attribute into the table or delete an attribute from the table, as we said earlier.

The system catalog discussed above will be used by all the operations in the MDBMS. The use of array index, compared to the use of pointer linked list structure, is judged to be superior; it saves a lot of time in searching the catalog tables and simplifies the implementation as well. However, while attributes are required to be unique within a user relation, same attributes names are permitted in different relations. Although this situation works fine for the formatted data because INGRES manages this kind of data and confusion will not arise, it creates problems for handling media data. In the MDBMS prototype, a separate relation, referred to as media relation in INGRES, is created for each media attribute. The name of this media relation is the same as the name of the media attribute. To avoid confusion and keep the media relations distinct when the same attribute name is used in more than one user defined relations, the names of these media relations are appended by suffixes, in this case numbers, corresponding to the unique system identifiers assigned to the user relations. [Po90]

When the user wants to create a relation, the procedure followed by the system each time might be different depending on the data types of the attributes of the new relation. If all the attributes of the new relation are formatted data type, then the entered information is transferred to INGRES via the *create table* command of SQL. On the other hand, in the existence of media data types the system asks INGRES to create a media relation for each media attribute, with the same attribute name in the Att_Array table and with a suffix. No duplication of attribute names is permitted within the same relation. The attributes of the media relation depends on the media type: the image media relations has the attributes image_id, file_id, description, height, width and depth; and the sound media

relation has sound_id, file_id, description, file_size, sampling_rate, encoding_technique, duration and resolution. In both media relations, the image_id and sound_id attributes are defined by the system internally and used by the system as a link between the main relation and these media relations. The second attributes (file_id) contain the exact path where the files, which include the registration data and raw data belong to each media, exist in the MDBMS system environment (like /n/virgo/mdbms/.../....snd). The description part is entered by the user in the natural language form, as explained in the previous section. The rest of the attributes are extracted from the media file and are used to display the image or to play the sound record. The media relation tables created by INGRES are shown in Figure 9. The detail information about the image media can be found in [Po90, Th88], and we will present the detail information about the sound media and its interface with the MDBMS system in the rest of the chapters, as the emphasis area of this thesis.

Picture

i_id	file_id	description	width	height	depth

Voice

s_id	file_id	description	size	samp_rate	encoding	duration	resolution

Figure 9. The Image and Sound Relational Tables

For the case of inserting tuples to the created relational tables, the insertion of the formatted data into the database can be done easily by using INGRES, but for the media data case, the procedure is different. The media files containing the data need to be transferred to the MDBMS environment before the insertion process. And when the system asks the user to enter the full path of the media data file, the user can give the address of the media file. Subsequently, the system creates an image_id or sound_id index depending on the next available row in the media relation, and stores the file_id as a whole path and the registration attributes by extracting from the media data file. Then, the user enters the description data for the media as stated before.

The retrieval operation is the most complex operation of the MDBMS prototype. When the system gets the required data from the user interactively, it builds an extended SQL query. If the query includes only the formatted data, it can be transferred directly to INGRES for processing, but for the media data a decomposition needs to be done. The system will separate the query into two parts: one part related to the formatted data and processed by INGRES, and the second part is done by using the information in the media relations. In the case of the existence of a media description in the condition part of the query (like officer with soft voice), the system uses Prolog to search the media data contents in the description attributes of the related media relation. Natural language descriptions are handled by means of a parser which transforms the description data into the Prolog predicates and literals to be deposited in a file named "imagei_image_facts" to be used by Prolog [Po90]. When the temporary tables are obtained after processing

each condition stated by the user, a join operation is done by the system to display the result table to the user.

The detail information about the catalog management, table creation and tuple insertion related to catalog tables can be found in [Pe90], and the retrieval operation in [Po90].

IV. SOUND MANAGEMENT

A. AN OVERVIEW OF SOUND CHARACTERISTICS

In this chapter, some of the basic characteristics of sound, which are used in the sound media data type, are discussed. As we stated earlier, the sound media data type includes sampling rate, encoding technique, resolution, size and duration as registration data, and these data are used to play the sound data in the system.

Sound can be defined as vibrations in a waveform in a conducting medium like air, water, wire etc. The sound we hear in the nature is in the analog waveform, which is a continuous wave. On the other hand, we store the sound in the computer in the digital form as a sequence of bits, as ones and zeros. The conversion of the sound in the analog form to digital form is accomplished by using a technique called *sampling*. Under certain conditions an analog sound signal can be completely represented by knowledge of its instantaneous values or *samples* equally spaced in time. This sampling process is done by using an analog-to-digital converter (*ADC*). The conversion of sampled energy levels into numerical quantities is known as *digitizing* [Sa88]. If the samples are sufficiently close together, in other words if we increase the rate of sampling, the digitized sound record is heard to be spatially continuous, like the original sound. We define the sampling rate in Hertz (Hz), for example 8000 samples per second corresponds to 8000 Hz or 8 Khz. On the other hand, when we increase the sampling rate, the size of the digitized sound

data that we store in the memory will also increase. So, there is a trade off between the quality of the digital sound record and the file size required to store the sound.

The encoding techniques are used to reduce the number of bits required to store the digitized sound. We will discuss the encoding techniques related to our system. During the encoding process, the systems filter and digitize the analog signal, analyze short segments of it, then encode prior to transmission or storage. The encoding technique, that we use in our system is *adaptive differential pulse code modulation* (*ADPCM*). This technique is included in the waveform encoding category, and is an extension of *pulse code modulation* (*PCM*). In PCM, the amplitude of the analog sound signal is sampled at a fixed rate and converted into digital information using an ADC. On the other hand, the *differential pulse code modulation* (*DPCM*) uses a different technique and stores the difference between the current value and the previous value of digitized amplitude, instead of the digitized amplitude for each sample. An improvement to differential PCM is achieved by ADPCM encoding technique, which predicts the next value by taking a few of the previous samples and extrapolating them. [Sa88]

The rest of the registration data is related to the storing of an encoded sound signal in the computer. The *resolution* is the number of bits used for each sample of sound signal, the *size* denotes the number of bits used to store the sound media, and the *duration* is the number of seconds to play the sound file and is based on the sampling rate and the rate of the playback device.

The discussion presented above is summarized from the predecessor M.S. Thesis by Sawyer [Sa88] and more detail information about general sound characteristics and techniques can be found there.

B. SOUND MEDIA OBJECT

The sound media object, as shown in Figure 2, is composed of three parts: registration data, raw data and description data. The registration data and the raw data parts are created in the PC environment, and the last part is created in the MDBMS during tuple insertion, as stated earlier.

When we record a sound, a file consisting of the registration data like file size, sample rate, encoding technique, duration and resolution, and the raw data, which is the bit string of the encoded digitized sound, is automatically created by the sound management application program. A unique file name is assigned to the created file, by using the recording date and time group.

We do not need to transfer the whole file to the MDBMS environment, since there is no sound support in the SUN workstations. We only send the registration data and the created unique file name of the sound record to the MDBMS system by using the *file transfer protocol* (***FTP***) in the Ethernet computer network.The registration data and the file name will be used in processing the queries.

During the MDBMS applications, when a user wants to listen to the sound record, a *remote play* command with the sound file name is sent to the PC sound management through the network. A *socket* abstract model, that will be discussed in the next section,

is used in the both ends of the network to establish the interface between the application programs and the network.

C. HARDWARE AND SOFTWARE REQUIREMENTS

The current MDBMS prototype is based on managing the sound media in an IBM compatible PC environment and accessing the sound media by using the sound interface through Ethernet, as stated earlier. The work related to the sound management prototype in the PC environment; as said before, has been done by a M.S. thesis student Sawyer in 1988. In this thesis, the previous work related to sound management has been modified. The network connection between the SUN workstation and PC, and the Sound Management Interface in the MDBMS prototype has been accomplished.

The sound media management in the MDBMS can be separated into three main parts according to the hardware and software requirements:

- Sound Management in IBM compatible PC environment
- PC - SUN Linkage
- Integration of Sound Management to the MDBMS in SUN

1. Sound Management Requirements in IBM Compatible PC

As stated earlier, the main work related to the sound management in the PC has been initiated by Sawyer in 1988 [Sa88]. In this thesis we modified and improved his program code for the current MDBMS prototype in a menu driven and user friendly fashion. We will talk about the implementation details in the next chapter.

We need the following hardware and software to accomplish the sound management in an IBM compatible PC environment (as shown in Figure 10):

- IBM compatible PC/AT with 20MB internal hard disk (80286 or higher)
- Antex VP620E PC compatible plug-in Digital Audio Processor board
- A speaker with a 1/4" standard jack
- A microphone with an audio amplifier and a 1/4" standard jack
- Microsoft C 5.0 with standard libraries
- Antex VP620ESE driver software
- MS-DOS version 3.0 or higher

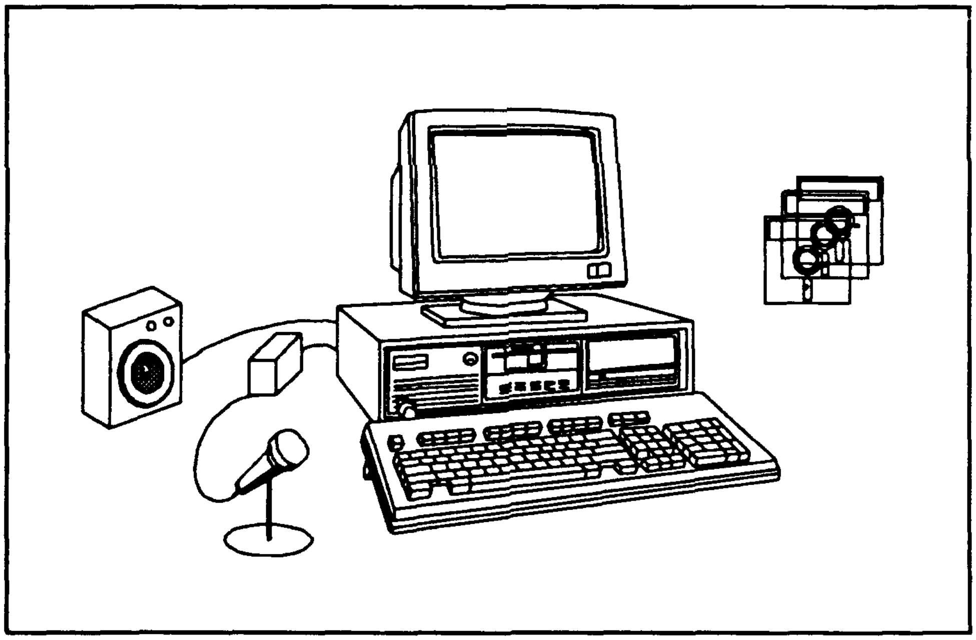

Figure 10. The Sound Management in IBM compatible PC

The Antex Electronic's Model VP620E PC compatible Digital Audio Processor board, together with other peripheral devices and software components, is the main

hardware unit used for sound management. This board converts real speech or music sounds to digital format for PC disk storage, and when one needs to playback, it is possible to convert the digital stored data back to the analog signal for playing with a speaker. These processes correspond to encoding and decoding of the sound signal, as we said earlier. The VP620E samples the audio waveform at either 8 or 16 Khz using ADPCM encoding technique.

The input sound signals can be captured in the system either by using a microphone together with an amplifier, (because the input signals must be at least 1 volt (RMS) in order to be accepted by the sound board as being above the threshold noise level) or the output port of an audio device like a cassette player, radio etc. Each sample is converted into an 8-bit digital number which is then compressed (encoded) using ADPCM by the *Digital Signal Processor* (*DSP*) chip, resulting in a 4-bit sound data sample. [Sa88]

The output process is the reverse of the capturing process. The 4-bit sound data is decoded back to 8-bit digital samples by the DSP chip, and the resulting samples then go to the speaker through amplifier stage. It is also possible to play sound in a background mode under DOS interrupt control, allowing the user to perform other operations in the MDBMS while a sound record is being played. This is accomplished by piping the sound data directly from the database to the DSP chip of the Antex sound board instead of RAM. [Sa88]

The Antex Audio Processor board comes with a software package and includes some basic routines, referred to as "commands", which can be called from high level languages like Basic, Pascal and 'C'. Once the VP620ESE driver software is activated,

these routines will run in the background independent of the status of an application program.

2. The PC - SUN Linkage Requirements

The task of connecting the PC to MDBMS in the SUN environment for the purpose of remotely interacting with the PC as a backend server from the MDBMS, was the main and critical issue in the beginning of this thesis work. First, we tried to establish a closed net by using a *null modem* between the SUN workstation and the PC, but the software (PROCOMM) we had for this connection did not fulfill our specifications, because this connection does not allow the PC and the SUN workstations to send and receive commands between them, and this connection requires to have the DOS in both ends. More over, it gives us difficulties of connecting multiple PC's to multiple SUN workstations. Then, we concentrated on using the Ethernet computer network, which is available and has already been used to connect the SUN workstations. The IBM compatible PC connection to the Ethernet was accomplished by using Excelan "The LAN Workplace" network software. The main communication between the SUN and PC was to play a given sound file name remotely in the PC. Consequently, we decided to use *remote shell* (***RSH***) command which is available in both the SUN and PC network software environment to execute the play file commands in the remote PC, but after a while we realized that Excelan network software package does not have RSH server capability, when RSH command is invoked from the SUN environment through Ethernet. Finally, we realized that we can use the *socket* model which is also available in both environment.

The socket interface model allows communication between an application program running on the system and the network. Specifically, this process is done in our PC between the running application program in the PC and the LAN Workplace TCP/IP Transport System software running on the EXOS 205T Intelligent Ethernet Controller board, which is already connected to the network. In the SUN environment, this process is automatically done by the 4.3 BSD UNIX operating system which is incorporated in SUN operating system by using available socket library routines as a feature of UNIX operating system. In both environments, the socket abstract model works in similar way.

Sockets are end points to which connections can be attached from the network and to which processes can be attached from the user in an application program. The socket system works with the TCP/IP, and has nothing to do with low level structure of the network [Ta88].The socket library provides for various functions such as reading, writing, accepting, etc. The socket functions are written in high level language 'C', so it is easy to integrate the socket routines to the application programs, which are already written in 'C' language. When a communication is desired between both ends of our system, the application program makes a request for the opening of a socket and all communication takes place through that socket. We use the sending and receiving sockets on both the PC and the MDBMS sides of the network. The detail information about the implementation of sockets will be presented in the next chapter.

We can list the necessary hardware and software for the PC - SUN linkage as following:

- Excelan EXOS 205T Intelligent Ethernet Controller board
- Excelan LAN (Local Area Network) Workplace Network Software for PC DOS TCP/IP Transport System
- Excelan LAN Workplace Network Software for PC DOS Socket Library Application Program Interface
- An Ethernet computer network
- Socket library routines for SunOS

3. The MDBMS Sound Management Integration Requirements

On the MDBMS side, we do not need that much hardware and software, because the system is already established in the SunOS environment, by using UNIX C. After the establishment of PC - SUN socket connection, all we need to do is to include the necessary socket routines in the MDBMS prototype, and a couple more routines into both the PC and the SUN systems to handle sound files and create the required sound media relations.

D. SOUND MANAGEMENT USER INTERFACE DESIGN

The user interface that we talk about in this section is related to the sound file creation and remote sound record playing; the part related to the MDBMS is already discussed in the previous chapter.

When a user wants to create a sound file, the *record* batch file is to be run in the PC. After running this batch file, the user gets the main menu as shown in Figure 11.

The first option is the *record sound* operation, which is used to create a sound record with the registration data from a standard input connection. This operation also

PC SOUND MANAGEMENT SYSTEM

1. **Record Sound**

2. **Play Sound**

0. **Exit**

Enter your choice number :

Figure 11. The main menu of PC Sound Management

creates a text file containing the sound file name and the registration data to be used in the MDBMS. The system asks the user to enter a file name for the text file, which will be transferred for setting up the registration data from PC to the SUN environment. The user does not need to know the long and complicated unique sound file name created by the system, this name is hidden from the user. Now, we can go through the user interface of the record sound operation in the following order (the *italics* represents the user's responses):

1 *<cr>*

==== RECORD SOUND ====

Please ENTER a file name to transfer the sound text file to SUN

yavuz *<cr>*

Please ENTER 0 (for sampling rate 8 Khz)
1 (for sampling rate 16 Khz)

0 *<cr>*

Recording in progress... Press << SPACE BAR >> to stop!

State=2 Error=0 Seconds=5.65 Overload=0

<< *SPACE BAR* >>

End of recording session!

Registration Data Values of the Recorded Sound File :

FILENAME : 08dh5255.snd
SIZE : 21584 bytes
SAMPLERATE : 8000 Hz
ENCODING : 1 (0:none, 1:ADPCM)
DURATION : 5.65 sec
RESOLUTION : 8 bits/sample

Please, press ENTER to continue...

<cr>

The next screen displays the main menu again, and the user can record another sound or play the recorded sound files or quit the program. The status information like *state*, *error*, *seconds* and *overload*, which are displayed on the screen during the sound recording, are defined by the Antex record function and can be found in the Antex VP620E user manual. For example, the parameter options for state status are: 0 for idle, 1 for playing, 2 for recording and 3 for waiting to play. The attributes of the registration data, *encoding* and *resolution* are previously set for this prototype, and they can not be changed by the user.

If the user wants to play the previously recorded sound file, he selects the *play sound* option :

2 <cr>

==== PLAY SOUND ====

Enter the file name to play (user defined) ?

yavuz *<cr>*

CURRENT FILE	: 08dh5255.snd
SIZE	: 21584 bytes
SAMPLERATE	: 8000 Hz
ENCODING	: 1 (0:none, 1:ADPCM)
DURATION	: 5.65 sec
RESOLUTION	: 8 bits/sample

Press << SPACE BAR >> to stop playing !

State=1 Error=0 Sec=5.65 Overload=0

**** E N D O F P L A Y ! ****

Please, press ENTER to continue...

Note that, to play a sound file from the PC end, a user only needs to give the text file name (e.g. yavuz) to the system. This, however, cannot be done from the SUN workstations. There the full sound file name (e.g. 08dh5255.snd) must be given. Such is the case as shown in the example at the end of this chapter. This has been done because only programs, not users, will invoke the command to play sound files from the SUN environment.

After recording the new sound files and listening them for verification purposes, if desired, the next steps are to exit the PC sound management system and to transfer each sound text file to the SUN workstation. We use *ftp* function to transfer these files in the Ethernet network, in the following order for the current system:

ftp virgo <cr>

Remote User Name : *mdbms<cr>*

Remote Password : *xxxxxx<cr>*

ftp> *cd snd <cr>*

ftp> *put yavuz <cr>*

ftp> *quit <cr>*

When the user is done with the sound file transfer, the next issue is to insert these sound data into the MDBMS databases. We have already talked about this in the previous chapter and will not be repeated here.

Now, we will discuss the *sound management integration interface* by using a program separate from the MDBMS, to be used for the test purposes for the PC - SUN linkage and the sound management integration. The routines of this program have been included in the main program of the MDBMS prototype. This program helps to promote better understanding of the sound management integration. However, these routines will not be invoked during the normal operations of the MDBMS prototype. They are used strictly for testing and debugging purposes. When this happens, the user runs the *sdemo* program in the SUN workstation. The system in turn asks for the name of the remote PC, which has the Antex VP620E digital audio processor board (for the current system, they are *pclum1* and *pclum2*). This is the PC that will be used for sound management, including the playing of the sound files remotely. Figure 12 shows the main menu after the PC has been selected.

To play the sound files stored in the PC equipped with the sound card, we use the socket abstract model that we introduced in the previous section. First, the PC end of the

SUN REMOTE SOUND PLAY MANAGEMENT

1. Play Sound File in PC

2. Display Sound File Header Information

0. Exit

Enter your choice number :

Figure 12. The main menu of the SUN Sound Management

system has to be ready to receive the play commands from the SUN workstation before the desired sound file can be played. We have a *receive.bat* file in the PC, which serves to receive the play commands and then play the sound files. This routine is embedded in a loop to allow the PC to continuously receive and play the sound files as desired by the user. On the other end of the system, in the SUN environment, we have the main menu in which the user can see the header information of a sound text file or play the sound file in the PC. As we stated earlier, this SDEMO program is used for test purposes, and we can not use the user defined sound file name as explained earlier.

If the user wants to play a sound file from the SUN end of the system, he sees the following interface :

2 <cr>

==== DISPLAY SOUND FILE HEADER INFO ====

Enter the 'USER DEFINED' sound text file name

yavuz *<cr>*

FILENAME : 08dh5255.snd
SIZE : 21584 bytes
SAMPLERATE : 8000 Hz
ENCODING : 1 (0:none, 1:ADPCM)
DURATION : 5.65 sec
RESOLUTION : 8 bits/sample

Please, press ENTER to continue...

<cr>

Now the system displays the main menu again (Figure 12), and if the user wants to play the sound file:

2 *<cr>*

==== PLAY SOUND IN PC ====

Enter the sound file name (xxxxxxxx.snd) to play in PC

08dh5255.snd *<cr>*

The system invokes the *send socket* routine with the given sound file name and sends the file name to the PC through the Ethernet. In case of any failure in the connection, the system displays a warning signal; otherwise the user can continue the process in the MDBMS without waiting for the end of playing. However, to play a second sound file one should wait for the end of playing; else the PC is not ready to receive again. On the PC end, the *receiver socket* receives the sound file name and writes it into a file. According to the order in the receive.bat file, the Antex Play function opens this file, reads the file name and plays the sound file. After that the PC is again ready to receive another sound file name from the send socket in SUN through the Ethernet.

In the following chapter, we will present the implementation detail of the sound management interface, which we discussed throughout this chapter.

V. IMPLEMENTATION OF SOUND MANAGEMENT INTEGRATION

In this chapter, we discuss the implementation details of the MDBMS sound management integration, one can find the program code related to this implementation discussion in the appendices of this thesis. First, we will present the implementation details of the sound management in PC, and then the sound management integration to the MDBMS prototype.

A. SOUND MANAGEMENT IN PC

The main issues in the PC sound management are recording sound and receiving the remote play commands from the MDBMS through Ethernet and playing the desired sound files. We use the following programs written in "C" language to manage the sound data in the PC:

- SND_REC.C (sound record)
- SND_STRU.C (sound structure)
- SND_ERRS.C (sound errors)
- SND_NAME.C (sound file name)
- PLAY.C (sound play)
- RECEIVER.C (receiver socket)
- RPLAY.C (remote sound play)

The main program for sound recording is SND_REC.C, and this program uses the modules: SND_STRU.C, SND_ERRS.C and SND_NAME.C besides the other library functions.

SND_STRU.C has the structure that represents the sound object whose attributes size, sampling rate, encoding, duration and resolution will be stored as a header file prefix to each recorded sound file.

SND_ERRS.C module contains a list of possible I/O error responses one might need during the processing of the programs.

SND_NAME.C creates the first four digits of the system created unique sound file name, by using the standard time parameters of GMT (Greenwich Mean Time). First digit stands for year, second one for month, third one for day and the last digit for hour. The last four digits are created by SND_REC.C program by using minute and second.

SND_REC.C records the sounds and plays them, if the user wants to. It creates two files, one with a unique file name consisting of the sound header information and the sound data, and the second one with a user defined name consisting of the unique file name and header information in a text format. There are two main functions in this program: *SSantex_record()* and *Ssantex_play()*.

SSantex_record() is the routine which records a sound file from a standard connection, and creates a sound object file with a unique name and a text file with a user defined name. This function is mainly a modified version of Antex VP620E record routine and uses the following functions: *Generate_filename(), ERROR store_snd_hdr(), ERROR store_ssun_hdr()* and *long FileSize()*. The **Generate_filename**() function produces

a unique 8-digit sound file name consisting of 1-digit each for year, month, day and hour by using SND_NAME.C external routine, and 2-digits each for minute and second, and adds ".snd" suffix to each file name. The **ERROR store_snd_hdr()** function stores the header information of the sound record into a designated file, followed by the sound data composed of bit strings. In other words it creates the sound object file. The **ERROR store_ssun_hdr()** function stores a unique sound file name and the header information into a designated user defined file as a text file to be sent to the SUN workstations through Ethernet to be used in the MDBMS. The **FileSize()** function determines the bit size of the recorded sound data file.

Ssantex_play() function is a modification of Antex VP620E play routine used to play a given sound file (with a user defined file name). This function uses the **Read_sfilename()** function to read the unique sound file name from the given sound text file, and the **Read_snd_hdr()** function to read the sound header information from the actual sound object file to play the record and display the header data.

PLAY.C plays a given sound file (in unique file name format xxxxxxxx.snd) and uses SSantex_play() and Read_snd_hdr() functions, discussed above.

RECEIVER.C is a modification of a receiver socket application program in the Excelan "The LAN WorkPlace" Network Software for PC DOS Socket Library Application Program Interface manual. We defined the number "2000" as a virtual port between this socket and the sending socket in the SUN workstation end. There are two options for this socket: to display the received data on the screen or to store the data into a designated file. We have chosen the second one, and now the receiver socket writes the

sound file name sent by the MDBMS through Ethernet, into a file named "**playfile**". We add this filename as an argument to store the sound file name, when the receiver socket program (*receiver -file playfile*) is called.

RPLAY.C plays the previously recorded sound files when they are requested by the MDBMS. This program first opens a designated file, which is given as an argument (for our case; *rplay playfile*), then fetches the sound file name written by the receiver socket, and plays the sound record. The functions in this program are: *SSantex_play()*, *Read_snd_hdr()*, *Read_filename()* and *Announce_play()*. We have talked about the first two functions already: the **Read_filename**() function which opens the given file (playfile) and reads the sound file name, and the **Announce_play**() function which is used to announce a message to warn the user in case of an error.

There are two batch files in the PC sound management system, other than the programs discussed above: *RECORD.BAT* and *RECEIVE.BAT*.

The **RECORD.BAT** file contains the following commands:

```
play 084m4325.snd
snd_rec
play 084m4434.snd
```

When the user enters *record* command, the system executes these commands in order. The play commands play the previously recorded interface announce files, like "Welcome to the sound management system" and "Thank you for using sound management system. Have a nice day". And between these announcements we have SND_REC.C program record and play sound, as discussed above.

The **RECEIVE.BAT** file contains the following commands:

```
:receive
     erase playfile
     receiver -file playfile
     rplay playfile
     erase playfile
     goto receive
```

When the user enters *receive* command, the system executes these commands in order and, after the execution of the last command, goes back to the first command and waits in a loop for the next action. There are two "erase playfile" commands, because the receiver socket can not write into a nonempty file. In the case of an existence of an unerased "playfile" file at the beginning of the loop, we put the first "erase playfile" command just before the receiver socket command. The system waits for the sound file name to be sent from the MDBMS end through Ethernet, after clearing the playfile. When a sound file name is sent from the MDBMS, the receiver socket receives the name and writes it into a file (playfile). Then with the next command (rplay playfile), the system opens this file and fetches the sound file name and plays the sound record. Finally, the system erases the playfile and waits to receive and play the next sound file name .

These are all the programs and routines that we use for the PC sound management, and the program code is included in Appendix A.

B. SOUND MANAGEMENT INTEGRATION INTO MDBMS

In the MDBMS environment, we need to read the sound text file which is sent from the PC end of the system, write these information into the sound relational tables (Figure 9), add description data as a third part of the sound media object during the tuple

insertion operation as stated earlier, and finally send the unique sound file name to the PC by using a sending socket to play a sound record.

We use the following programs to illustrate how the sound management is integrated into the MDBMS:

- SDEMO.C (SUN sound demo)
- SND_STR.C (sound structure)

SDEMO.C is the main program and uses the module SND_STR.C. The module **SND_STR.C** is different from the SND_STRU.C, because we added the unique sound file name attribute in addition to the header information in this structure.

SDEMO.C sends a sound file name to the PC to play sound record by using a sending socket and displays the information in the sound text file. This program first asks for the remote PC name, which has sound processing capability, to play sound records before displaying the main menu. There are two main functions in this program: *Sendsocket()* and *Snd_inforead()*.

SendSocket() is a sending socket function and is a modification of the "Internet domain stream connection" program in the SUN Microsystem Network Programming manual. It sends the remote sound file name to the given PC through a virtual port (number: 2000). In the case of failure in the connection, the system displays an error message, otherwise the user can continue the other applications in the MDBMS.

Snd_inforead() reads the unique sound file name and header information from a given sound text file, which is sent from the PC to the SUN workstation.

The functions in the SDEMO.C program, SendSocket() and Snd_inforead, are included in the main program of the MDBMS prototype to send the sound file names to the PC and to store the header information into the sound relational tables during tuple insertion. The program code related to the SUN sound management is given in Appendix B.

VI. SUMMARY AND CONCLUSIONS

A. REVIEW OF THESIS

The current MDBMS prototype at the Computer Science Department of the Naval Postgraduate School can let the user to capture, store, manage, retrieve and present the image and sound media data together with standard data like numerics and alphanumerics. The system, although conceptually using a SQL-like query interface to process the operations in the MDBMS, demands interactively the necessary data from a user in a user friendly way, without any formatted SQL statement. In this way, the users do not need to have wide and deep background or knowledgeable about the database systems.

In the MDBMS, we need to deal with both the standard data and the multimedia data together. We use the INGRES DBMS to manage the standard data; for the multimedia data we need to establish our own database structure to handle the media data. A main issue on processing the multimedia data is how to address the contents of the media data. Handling the content search issue is not possible with the conventional methods used in the current DBMSs, so we need to use an abstract data type concept to let us handle the media data. And also we need to have some parameters together with the raw media data to display the media data, e.g. size, pixel depth, resolution, encoding and colormap for the image data; and size, sampling rate, encoding, duration and resolution for sound data. The media data used to implement abstract data type concept is composed of the following parts: registration data, raw data and description data. T.e

registration data part includes the parameters for the interpretation or display of the media data, has a fixed format, because the formats or field lengths of the parameters are known. The raw data is in a bit string format like a pixel matrix of an image, or the bit string representation of a sound record. And the description data, that describes the object content of the raw data and is used for content search purposes, is entered by the users in natural language form.

A parser was constructed for processing natural language description data. A dictionary or lexicon is prepared for the MDBMS, and the user is restricted to use the words or phrases as defined in this dictionary to write the description data. The parser transforms the description data entered by the user into a set of predicates and literals suitable for Prolog processing. These predicates state a fact about the content in the media object. When the user enters a query by using a media content description like "officer with glasses", the system calls the parser to perform a content search throughout the media data descriptions.

Three M.S. thesis students worked on the design and implementation of the current MDBMS prototype. After the collective work on the design part, the detail design and implementation for table creation and tuple insertion with catalog management were done by Pei [Pe90], and the design and implementation for the retrieval operation were done by Pongswuan [Po90]. In this thesis, we accomplished the sound management integration into the system, which includes the storage and management of the sound media data using an IBM compatible personal computer, the connection between the MDBMS and the PC, and the integration into the MDBMS.

The user can record and play sound data in the PC. During the sound recording process two different files are created: one with the registration data and the bit string of sound data with a system created unique file name, and the other one a text file containing the unique sound file name and the registration data belongs to the sound record with a user defined file name. The description part of the sound media object file is created in the SUN workstation environment during the tuple insertion process. Only the second file , sound text file, is transferred from the PC to the MDBMS environment to be stored in the sound media relations. When the user wants to play sound, the system only sends the unique sound file name to the PC through the local area network, Ethernet. There are two application systems working simultaneously, one in the MDBMS for sending a sound file name and the second one in the PC, waiting to receive the sound file name sent from the MDBMS for playing. We used the socket abstract model through Ethernet to accomplish this connection between these two application systems.

B. APPLICATION AREAS

We can imagine many application areas for the current MDBMS prototype with image and sound processing capability. We can give the following application scenarios as an example:

Electronic Warfare Training: We can store the parameters of the electronic devices like radars, the picture of the intercepted signals and the sound record of these signals. The EW operator can guess the name of the radar and parameters by looking at the picture of the signal and listening the sound record.

Ship and Weapon Database: We can store the formatted information about the ships and weapons like name, type, length, number, displacement, power, range etc., and the pictures and sound record if it is available and needed.

Personnel Database: We can store the information about the personnel like name, rank, job, age, weight, height, salary, marital status etc. with the photographs, fingerprints and voice prints of the personnel.

News Archive Database: We can store the pictures and the narrative explanation of an event or accident, or a ceremony etc.

C. FUTURE WORKING AREAS

Currently not all the database operations of the MDBMS prototype completed. More work needs to be done for the modification and deletion operations in the databases, the enhancement of the querying capability like nested queries, and more effective help utilities.

The system can be improved with a more user friendly interface, similar to the modern graphical interfaces like using mouse clicks in colorful windows instead of typing after each prompt.

At this time three more M.S. thesis students Aygun, Peabody and Stewart [Ay91,Pe91,St91] are working on these areas.

LIST OF REFERENCES

[Ay91] Aygun, H., *Design and Implementation of a Multimedia DBMS: Complex Query Processing*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California. (in preparation)

[Be86] Bertino, E., Gibbs, S., Rabitti, F., Thanos, C. and Tsichritzis, D., "A Multimedia Document Server," in Proc. 6th Japanese Advanced Database Symposium (Tokyo, August 1986), *Information Processing Society of Japan*, pp.123-134, 1986.

[BRG88] Bertino, E., Rabitti, F. and Gibbs, S., "Query Processing in a Multimedia Document System," *ACM Trans. on Office Information Systems*, v.6,no.1, pp.1-41, January 1988.

[Ch86] Chrisodoulakis, S., Theodoridou, M., Ho, F., Papa, M. and Pathria, A., "Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A model and a System," *ACM Trans. on Office Information Systems*, v.4, no.4, pp. 345-383, October 1986.

[Du90] Dulle, J., *The Scope of Descriptive Captions for Use in a Multimedia Database System*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California, June 1990.

[EN89] Elmasri, R. and Navathe, S.B., *Fundamentals of Database Systems*, pp. 504, The Benjamin/Cummings Publishing Company, Inc., 1989.

[KKS87] Kosaka, K., Kajitani, K. and Satoh, M., "An Experimental Mixed-Object Database System," in Proc IEEE Cs Office Automation Symposium (Gaithersburg, MD, April 1987), IEEE CS Press, order no. 770, pp. 57-66, Washington, 1987.

[LM88] Lum, V.Y. and Meyer-Wegener, K., "A Conceptual Design for a Multimedia DBMS for Advanced Applications," Report no. NPS-52-88-025, Naval Postgraduate School, Department of Computer Science, Monterey, California, August 1988.

[LM89] Lum, V.Y. and Meyer-Wegener, K., "A Multimedia Database Management System Supporting Content Search in Media Data," Report no. NPS-52-89-

020, Naval Postgraduate School, Department of Computer Science, Monterey, California, March 1989.

[LM90] Lum, V.Y. and Meyer-Wegener, K., "An Architecture for a Multimedia Database Management System supporting Content Search," in Proceedings of the international Conference on Computing and Information (ICCI'90), Niagara Falls, Canada, May 23-26 1990.

[Lo88] Lockemann, P.C., "Multimedia Databases : A Paradigm and Architecture," Report no. Nps-52-88-047, Naval Postgraduate School, Department of Computer Science, Monterey, California, September 1988.

[MLW88] Meyer-Wegener, K., Lum, V.Y. and Wu, C.T., "Managing Multimedia Data - An Exploration," Report no. NPS-52-88-010, Naval Postgraduate School, Department of Computer Science, Monterey, California, March 1988. Also published in *Visual Database Systems*, Kunii, T.L., pp. 497-523, The North-Holland Publishing Company, Inc., 1989.

[Pe90] Pei, S., *Design and Implementation of a Multimedia DBMS: Catalog Management, Table Creation and Data Insertion*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California, December 1990.

[Pe91] Peabody, C., *Design and Implementation of a Multimedia DBMS: Graphical User Interface Design and Implementation*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California. (in preparation)

[Po90] Pongsuwan, W., *Design and Implementation of a Multimedia DBMS: Retrieval Management*, Master's Thesis, Naval Postgraduate School, Department of Computer Sciawce, Monterey, California, September 1990.

[Sa88] Sawyer, G.R., *Managing Sound in a Relational Multimedia Database System*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California, December 1988.

[St91] Stewart, R., *Design and Implementation of a Multimedia DBMS: Modification and Deletion*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California. (in preparation)

[Ta88] Tanenbaum, A.S., *Computer Networks*, Prentice-Hall, Inc.,(2nd Edition), 1988.

[Th88] Thomas, C.A., *A Program Interface Prototype for a Multimedia Database Incorporating Images*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California, December 1988.

[WNT89] Wu, C.T., Nardi, P., Turner, H., Antonopoulos D., "ARGOS Next Generation Shipboard Information Management System," Report no. NPS-52-90-006, Naval Postgraduate School, Department of Computer Science, Monterey, California, December 1989.

[WK87] Woelk, D. and Kim, W., "Multimedia Management in an Object-Oriented Database System," Proc. 13th International Conference on VLDB, Brington, England, September 1987.

APPENDIX A - PC SOUND MANAGEMENT PROGRAM CODE

The following program code is either created or modified from the code written by Sawyer [Sa88] and Antex VP620ESE driver software, for the implementation of sound management in the PC.

```
/*************************** SND_REC.C ********************************
* Title       : SOUND RECORD AND MANAGEMENT IN PC
* Author      : Yavuz Vural ATILA
* Rank        : LTJG Turkish Navy
* Advisor     : Prof. Vincent Y. LUM
* Date        : 27 August 1990
* Revised     : 18 September 1990
* Description : This program is designed for ANTEX VP620E sound
*               processing card, and it records a sound data file from a
*               standard input connection (i.e. microphone, tape recorder)
*               It creates an header file automatically, which includes
*               file size(Bytes), sampling rate (Hz), encoding technique,
*               duration (Sec), and resolution (Bits/sample).
*               And also it plays the sound file which is recorded
*               previously.
*************************  *********************************************/

#include <stdio.h>
#include <sys/types.h>
#include <time.h>
#include "snd_stru.c"
#include "snd_errs.c"
#include "snd_name.c"

#define  NAME_LENGTH 13
#define  ERROR_FREE 0
#define  SOUND_ERROR -1

#define BEGIN  1;            /*ANTEX VP620E commands*/
#define SETREC 2;
#define START  4;
```

```
#define STOP   5;
#define STATUS 6;
#define PLAY   8;
#define END    9;

int VP620 ();

/* ----------------------- Generate A New File Name ------------------------
   Produce a unique 8-digit filename for the recording composed of
   1-digit each for year, month,day & hour, and 2-digits each for minute and
   second. Each sound object can be identified by the ".snd" suffix.
   ---------------------------------------------------------------------- */

Generate_filename(sound_filename)
   char  *sound_filename;          /* unique sound file name */

{
   char *p;
   struct tm *t;
   time_t current_time;

   current_time = time(NULL);

   t = gmtime(&current_time);

   sprintf(sound_filename,"%1c%1c%1c%1c%2d%2d.%s",
       YR(t->tm_year),MN(t->tm_mon), DAY(t->tm_mday), HR(t->tm_hour),
       t->tm_min, t->tm_sec,"snd");

   sound_filename[NAME_LENGTH-1] ='\0';

   for (p=sound_filename;*p;p++)
       if (*p==' ')
         *p = '0';
   return ERROR_FREE;

}

/* ---------------------- Store Sound Header and Data ---------------------
   This function stores a header info into a designated  file,then reads
   the recorded sound file, buffers the data,then writes the buffer into
   the designated file following the header.
   ---------------------------------------------------------------------- */
```

```
ERROR store_snd_hdr(fname,r,temp_file)
  char    *fname;              /* given unique sound filename */
  struct SND_HDR  r;             /* sound object record */
  char    *temp_file;
  {
    FILE  *f,*fg;
    char  *buf[500];            /* input/out buffer */
    int   num;


    if ((f = fopen(fname,"wb")) == NULL)        /* open for writing */
     return(WOPEN);

    if ((fg = fopen(temp_file,"rb")) == NULL)     /* open for reading */
     return(ROPEN);

    num = 1;                              /* only one header */


    /* ***** write the header into the predesignated output file   */
    if (fwrite(&r, sizeof(struct SND_HDR), 1, f) < num )
     return (WRITE);                       /*   write error */


    while (!feof(fg))
     {
     if (fread(buf,500,1,fg) < 0 )          /* load buffer */
       return(READ);

      /* ***** append data from sound data buffer */
     if (fwrite(buf, 500, 1, f) < num )        /* write buffer */
       return (WRITE);
     }

     if (fclose(f) != 0)
     return(WCLOSE);                       /*  close error */

    if (fclose(fg) != 0)
     return(WCLOSE);

   return(OK);
  }
```

```
/* --------------------- Store SUN Sound Header ----------------------
   This function stores a header information into a designated  file
   as a text file, to send to the SUN for using in the MDBMS.
   ------------------------------------------------------------------ */

ERROR store_ssun_hdr(fname,r,sfname)
  char    *fname;                    /* given sound file */
  struct SND_HDR r;                    /* sound object record */

  {
    FILE  *f;

    if ((f = fopen(fname,"w")) == NULL)       /* open for writing */
     return(WOPEN);

    fprintf(f,"%s\0",sfname);          /* store the header info */
    fprintf(f,"\n%ld\0",r.s_size);     /* into a text file */
    fprintf(f,"\n%d\0",r.s_samplrate);
    fprintf(f,"\n%d\0",r.s_encoding);
    fprintf(f,"\n%f\0",r.s_duration);
    fprintf(f,"\n%d\0\n",r.s_resolution);

    if (fclose(f) != 0)
     return(WCLOSE);                  /*  close error */


   return(OK);
  }

/* ------------------ Determine Size of DATA-ONLY File --------------- */

long FileSize(i_file)

     char  *i_file;                    /* input file */
{
     FILE  *f;
     long int f_size;

     if ((f = fopen(i_file, "rb")) == NULL)    /* open file */
         displayerr(ROPEN);

     if (fseek(f,0L,2) != 0)      /*  set position rel. to end */
        return EOF;
```

```
        f_size = ftell(f);


        if (fclose(f) != 0)                /* close file */
           displayerr(RCLOSE);

        return f_size;
}

/********************** ANTEX_RECORD FUNCTION **********************
   This function records a sound file from a standard connection and
   creates two output files; one sound object file and the other one
   text file contains the registration data and the unique sound file name.
 *******************************************************************/

Ssantex_record(filename,sfilename)

char   *filename;          /* sound object file name */
char   *sfilename;         /* sound text file name */
{

   int port,useint;
   int vpfunction,samplerate;
   int state,error,sec,hundsec,overload;
   int monitor = 1;                 /* record monitor always on */

   long int sz;
   int  srate,sencode,sresol;
   float sdur;
   char c;

   int   *pi,               /* storage allocation stmts...unused */
       i = 0;
   char  *newname;

   ERROR  err;
   struct SND_HDR hdr;

   /* ***** Executable statements ***** */

   generate_filename(filename);
   puts("Please ENTER a filename to transfer the sound text file to SUN");
```

```
gets(sfilename);

printf("Please ENTER  0  for sampling rate 8 Khz\n");
printf("              1  for sampling rate 16 Khz\n");
scanf("%d",&samplerate);
c = getchar();

vpfunction = BEGIN;  /* vpbegin */
port = 0x280;        /* use default IO address */
useint = 2;          /* use interrupt 2 */
VP620(&vpfunction,&useint,&port);/* wake-up call to driver */


vpfunction = SETREC; /* vpsetrec */
sencode = 1;              /* ANTEX recording w/8bit resolution */

newname = "temp.snd";
VP620(&vpfunction,&monitor,&samplerate,newname);

vpfunction = START;  /* vpstart */
VP620(&vpfunction);

printf("\nRecording in progress...Press  << SPACE BAR >> to stop!\n\n");

vpfunction = STATUS;/* vpstatus */
do
   {
   VP620(&vpfunction,&overload,&hundsec,&sec,&error,&state);
   printf(" State=%d  Error=%d  Seconds=%d.%02d  Overload=%d\r",
        state,error,sec,hundsec,overload);

   }
while(!kbhit() & state!=3);


/* These statements always required to close the file */
vpfunction = STOP;    /* vpstop */
VP620(&vpfunction);

printf("\n");
printf("\n\nEnd of recording session! \n");

vpfunction = END;     /* vpend */
```

```
    VP620(&vpfunction);

 /* Update the header fields now that the file has been recorded */

    sz = FileSize(newname) + sizeof(struct SND_HDR);
    hdr.s_size = sz;

    if (samplerate == 0)
       srate = 8000;
    else
       srate = 16000;

    hdr.s_samplrate = srate;

    hdr.s_encoding = sencode;               /* ADPCM code */

    sdur = (sec + ((float) hundsec / 100));
    hdr.s_duration = sdur;

    if (sencode == 1)
       hdr.s_resolution = 8;                /* set #bits-per-sec */
    else
       hdr.s_resolution= 0;
    /*  ***** header values of recorded sound file */
       printf("\nRegistration Data Values of the Recorded Sound File    :");
       printf("\n-----------------------------------------------------\n");
       printf("\nFILENAME     : %s\n",filename);
       printf("\n\n\tSIZE       : %ld bytes\n",hdr.s_size);
       printf("\tSAMPLERATE: %d Hz\n",hdr.s_samplrate);
       printf("\tENCODING  : %d (0:none, 1:ADPCM)\n",hdr.s_encoding);
       printf("\tDURATION  : %5f sec\n",hdr.s_duration);
       printf("\tRESOLUTION: %d bits/sample\n\n",hdr.s_resolution);

 /* ***** store header and data into designated file  */
    if ((err = store_snd_hdr(filename,hdr,newname)) != OK)
       displayerr(err);

    if ((err = store_ssun_hdr(sfilename,hdr,filename)) != OK)
       displayerr(err);

 printf("\n\t\tPlease, press ENTER to continue...");

 }
```

```
/* ----------------------- Read Sound Header --------------------------
   This function reads a header from a designated  file,and returns
   the header info to the caller with the variable fields updated.
   ------------------------------------------------------------------- */
Read_snd_hdr(fname,h)
   char    *fname;                        /* given sound file */
   struct SND_HDR  *h;                        /* sound object header */
   {
     FILE  *f;
     int   num = 1;                       /* only one header */

     if ((f = fopen(fname,"rb")) == NULL)      /* open for reading */
      {
      displayerr(ROPEN);                  /* open file error */
      return SOUND_ERROR;
      }

     /* ***** read the header info from the predesignated input file    */
     if (fread(h, sizeof(struct SND_HDR), 1, f) < num )
      {
      displayerr(READ);                   /*   read error */
      return SOUND_ERROR;
      }

     if (fclose(f) != 0)
      {
      displayerr(WCLOSE);                  /*  close file error */
      return SOUND_ERROR;
      }

    return ERROR_FREE;
   }

/* ----------------------- Read sound file name -------------------------- */

 Read_sfilename(stfilename,sfname)

      char  *stfilename;              /* given text sound file name */
      char  *sfname;                  /* actual sound file name */

   {
     FILE  *f;
```

```
    if ((f = fopen(stfilename,"r")) == NULL)        /* open for reading */
       {
        displayerr(ROPEN);
        return SOUND_ERROR;
       }

     /* ***** read the header from the predesignated input file   */
          fscanf(f,"%s",sfname);

      fclose(f);
return;
}

/********************** ANTEX_PLAY FUNCTION **************************
 This function plays the given sound file.
*********************************************************************/

SSantex_play(stfilename)

   char  *stfilename;                  /* given text sound file */

{
  int port,useint;                  /*  declarations */
  int vpfunction,samplerate;
  int state,error,sec,hundsec,overload;
  int srate;
  char filename[13];

  struct SND_HDR hdr;

  int monitor = 1;                  /*  record monitor always on */

  ERROR  err;


  /*  ***** Executable Statements ***** */

  Read_sfilename(stfilename,filename);

  if ((read_snd_hdr(filename,&hdr)) == 0)
    {
     /*  ***** display header values for information */
```

```
printf("\nCURRENT FILE : %s\n",filename);
printf("\n\tSIZE       : %ld bytes\n",hdr.s_size);
printf("\tSAMPLERATE: %d Hz\n",hdr.s_samplrate);
printf("\tENCODING  : %d (0:none, 1:ADPCM)\n",hdr.s_encoding);
printf("\tDURATION  : %5f sec\n",hdr.s_duration);
printf("\tRESOLUTION: %d bits/sample\n\n",hdr.s_resolution);

vpfunction = BEGIN;              /* alert to the driver */

port = 0x280;
useint = 2;
VP620 (&vpfunction, &useint, &port);


if (hdr.s_samplrate == 8000)
     srate = 0;
else
     srate = 1;

vpfunction = PLAY;
VP620 (&vpfunction, &srate, filename);  /*  open file */

vpfunction = STATUS;

printf("\n\tPlease, press << SPACE BAR >> to stop playing !\n\n\n");

do {
  VP620 (&vpfunction, &overload, &hundsec, &sec, &error, &state);
  printf("  State= %d   Error = %d  Sec = %d.%d  Overload = %d \r",
     state, error, sec, hundsec, overload);
  }

while (!kbhit() & state != 3);

printf("\n\n\n\t**** E N D   O F   P L A Y ! ****\n");
printf("\n\nPress ENTER to continue...");

/* these statements always required to close the file */
vpfunction = STOP;
VP620 (&vpfunction);

vpfunction = END;
VP620 (&vpfunction);
```

```
        return ERROR_FREE;
    }
  else
    {
    displayerr(READ);                    /*file read error*/
    return SOUND_ERROR;
    }
  }

/* ************************** CLEAR SCREEN ************************** */
clr_scr()
{
        putchar('\033');
        putchar('[');
        putchar('H');
        putchar('\033');
        putchar('[');
        putchar('J');
 }

/****************************** M A I N ********************************/

main()

{
        char filename[NAME_LENGTH];
        char sfilename[NAME_LENGTH];
        char answer,a;

        while (answer != '0') {
             answer = 0;
             while (answer < '0' || answer > '2') {
        clr_scr();
        printf("\n\n\t\t\tPC SOUND MANAGEMENT SYSTEM\n");
        printf("\t==============================================");
        printf("\n\t\t1. Record Sound");
        printf("\n\t\t2. Play Sound");
        printf("\n\t\t0. Exit\n");
        printf("\t==============================================\n");
        printf("\n\tEnter your choice number :  ");

        answer = getchar();
```

```
    while ((a=getchar()) != '\n');
        printf("\nYour answer is %c\n", answer);

    switch(answer)
 {

        case '1' :
            clr_scr();
            printf("\t\t==== RECORD SOUND ==== \n");
            SSantex_record(filename,sfilename);
            a= getchar();
            break;

        case '2' :
            clr_scr();
            printf("\t\t==== PLAY SOUND ==== \n");
            puts("Enter the file name to play (user defined) ?");
            gets(sfilename);
            SSantex_play(sfilename);
            a= getchar();
            break;

        case '0' :
            clr_scr();
            printf("\n\n\n\t\t\t************************ \n");
            printf("\t\t\t====== THANK YOU ====== \n");
            printf("\n\t\t\t==== HAVE A NICE DAY ==== \n");
            printf("\t\t\t************************ \n");
            break;

        } /* end switch */
    }  /* end while ans not in 0..3 */
  getchar();
  } /* end while ans not '0' */
} /* end main */

/*************************************************************************/
```

```
/****************************** RPLAY.C ********************************
* Author      : Yavuz Vural ATILA
* Rank        : LTJG, Turkish Navy
* Advisor     : Prof. Vincent Y. LUM
* Date        : 27 August 1990
* Revised     : 17 September 1990
* Description : This program is designed for the ANTEX VP620E, to play
*               the previously recorded sound files, when requested by
*               a remote host.
*               The filename , which is sent by the remote host (SUN),
*               first is stored into a file and then this program reads
*               that file ("playfile") and fetches the sound file name
*               and plays it.
**********************************************************************/

#include <stdio.h>
#include <sys/types.h>
#include <time.h>
#include "snd_stru.c"
#include "snd_errs.c"

#define  ERROR_FREE 0
#define  SOUND_ERROR -1

#define BEGIN  1;         /* ANTEX VP620E commands */
#define SETREC 2;
#define START  4;
#define STOP   5;
#define STATUS 6;
#define PLAY   8;
#define END    9;

char *no_file = "08hh0053.snd";   /* previously recorded announce files */
char *not_read= "08hh0214.snd";
/* The annouces in these files are: "There is no such a file in memory"
                                    "The given file can not be read" */

/* ************************ PLAY ANNOUNCE FILES **********************
   This function plays the previously recorded warning announce files.
   ****************************************************************** */

Announce_play(filename)
```

```
    char  *filename;                    /* sound filename */

{

/*  declarations */
    int port,useint;
    int vpfunction,samplerate;
    int state,error,sec,hundsec,overload;

    int monitor = 1;               /*  record monitor always on */

    int  srate;

    ERROR  err;
    struct SND_HDR  hdr;


    /*  ***** Executable Statements ***** */

    if ((read_snd_hdr(filename,&hdr)) == 0)
    {

        vpfunction = BEGIN;             /* alert to the driver */

        port = 0x280;
        useint = 2;
        VP620 (&vpfunction, &useint, &port);


        if (hdr.s_samplrate == 8000)
             srate = 0;
        else
             srate = 1;

        vpfunction = PLAY;
        VP620 (&vpfunction, &srate, filename);  /*  open file */

        vpfunction = STATUS;

    do {
      VP620 (&vpfunction, &overload, &hundsec, &sec, &error, &state);
      }
    while (!kbhit() & state != 3);
```

```
  /*  these statements always required to close the file */

      vpfunction = STOP;
      VP620 (&vpfunction);

      vpfunction = END;
      VP620 (&vpfunction);

      return ERROR_FREE;
    }
else
  {
  displayerr(READ);
  return SOUND_ERROR;
  }
}

/************************ READ FILENAME ******************************
  This function reads a filename from the "playfile" file, which is created
  by the receiver socket.
*****************************************************************************/

Read_filename(filename,rpfname)

  char  *filename;              /* output */
  char  *rpfname;                /*remote playfile name*/

  {
    FILE  *f;


    if ((f = fopen(rpfname,"rb")) == NULL)       /* open for reading */
     {
     displayerr(ROPEN);
     return SOUND_ERROR;
     }


    /* ***** read the filename from the predesignated input file    */
    if (fread(filename,12,1,f) < 1 )
     {
     displayerr(READ);                  /*   read error */
     return SOUND_ERROR;
```

```
    }

   if (fclose(f) != 0)
    {
    displayerr(WCLOSE);                /*  close error */
    return SOUND_ERROR;
    }

  return ERROR_FREE;
 }


/*********************** READ SOUND HEADER **************************
  This function reads a header from a designated  file,and returns
  the header to the caller with the variousfields updated.
*********************************************************************/

Read_snd_hdr(filename,h)

  char    *filename;              /* output */
  struct SND_HDR  *h;               /* sound object record */

  {
   FILE  *f;
   int   num = 1;            /* only one header */


   if ((f = fopen(filename,"rb")) == NULL)       /* open for reading */

    {
    displayerr(ROPEN);
    ann_play(no_file);
    return SOUND_ERROR;
    }


   /* ***** read the header from the predesignated input file   */

   if (fread(h, sizeof(struct SND_HDR), 1, f) < num )

    {
    displayerr(READ);                 /*   read error */
```

```
      ann_play(not_read);
      return SOUND_ERROR;
      }


    if (fclose(f) != 0)

      {
      displayerr(WCLOSE);                 /*  close error */
      return SOUND_ERROR;
      }

   return ERROR_FREE;

  }

/********************** ANTEX_PLAY FUNCTION *************************
 This is the actual function that plays the sound.Its input is a filename.
 A successful play will return a '0' to the caller.  Failure will return
 an error message.
*********************************************************************/

SSantex_play(filename)

char  *filename;                   /* primary input file */

{

/*  declarations */
   int port,useint;
   int vpfunction,samplerate;
   int state,error,sec,hundsec,overload;

   int monitor = 1;                /*  record monitor always on */

   int  srate;

   ERROR  err;
   struct SND_HDR  hdr;


   /*  ***** Executable Statements ***** */
```

```
if ((read_snd_hdr(filename,&hdr)) == 0)
{

/*  ***** read header values to set parameters */

    printf("\n\nCURRENT FILE : %s\n",filename);
    printf("\n\tSIZE       : %ld bytes\n",hdr.s_size);
    printf("\tSAMPLERATE: %d Hz\n",hdr.s_samplrate);
    printf("\tENCODING  : %d (0:none, 1:ADPCM)\n",hdr.s_encoding);
    printf("\tDURATION  : %5f sec\n",hdr.s_duration);
    printf("\tRESOLUTION: %d bits/sample\n\n",hdr.s_resolution);


    vpfunction = BEGIN;             /* alert to the driver */

    port = 0x280;
    useint = 2;
    VP620 (&vpfunction, &useint, &port);

    if (hdr.s_samplrate == 8000)
         srate = 0;
    else
         srate = 1;

    vpfunction = PLAY;
    VP620 (&vpfunction, &srate, filename); /*  open file */
    vpfunction = STATUS;
    printf("\n\tPlease press  << SPACE BAR >> to stop playing...\n\n");

do {
   VP620 (&vpfunction, &overload, &hundsec, &sec, &error, &state);
   printf("  State= %d   Error = %d  Sec = %d.%d  Overload = %d \r",
      state, error, sec, hundsec, overload);
   }
while (!kbhit() & state != 3);

/*  these statements always required to close the file */
    vpfunction = STOP;
    VP620 (&vpfunction);

    vpfunction = END;
    VP620 (&vpfunction);
```

```
      return ERROR_FREE;
    }

else
    {
    displayerr(READ);
    return SOUND_ERROR;
    }
}

/****************************** M A I N ********************************/

main(argc,argv)              /* receives the filename , which sound */
 int argc;                  /* filename is already written inside  */
 char *argv[];              /* by PC receiver socket program.     */
{
   char filename[12];

      read_filename(filename,argv[1]);/*read the filename from external*/
      SSantex_play(filename);        /*file*/

} /* end of main */
/*************************************************************************/
```

```
/************************** PLAY.C *************************************
* Title        : PLAY SOUND FILE IN PC
* Author       : Yavuz Vural ATILA
* Rank         : LTJG Turkish Navy
* Advisor      : Prof. Vincent Y. LUM
* Date         : 27 August 1990
* Revised      : 17 September 1990
* Description : This program plays a given sound file (in xxxxxxxx.snd
*               format) by using the ANTEX VP620E sound processor board.
***********************************************************************/

#include <stdio.h>
#include <sys/types.h>
#include <time.h>
#include "snd_stru.c"
#include "snd_errs.c"

#define  ERROR_FREE 0
#define  SOUND_ERROR -1

#define BEGIN  1;          /* ANTEX VP620E commands */
#define SETREC 2;
#define START  4;
#define STOP   5;
#define STATUS 6;
#define PLAY   8;
#define END    9;


int VP620 ();


/* --------------------------- Read Sound Header ---------------------------
   This function reads a header from a designated  file,and returns
   the header info to the caller with the variable fields updated.
   -------------------------------------------------------------------------- */

Read_snd_hdr(fname,h)

  char    *fname;                    /* given sound file */
  struct SND_HDR  *h;                 /* sound object header */

  {
```

```
      FILE  *f;
      int   num = 1;                          /* only one header */

      if ((f = fopen(fname,"rb")) == NULL)     /* open for reading */

       {
       displayerr(ROPEN);                     /* open file error */
       return SOUND_ERROR;
       }

      /* ***** read the header info from the predesignated input file   */

      if (fread(h, sizeof(struct SND_HDR), 1, f) < num )
       {
       displayerr(READ);                      /*   read error */
       return SOUND_ERROR;
       }


      if (fclose(f) != 0)
       {
       displayerr(WCLOSE);                    /*  close file error */
       return SOUND_ERROR;
       }

    return ERROR_FREE;
    }


/********************* ANTEX_PLAY FUNCTION *************************
 This function plays the given sound file.
*********************************************************************/

SSantex_play(filename)

char  *filename;                    /* sound filename */

{

/*  declarations */
   int port,useint;
   int vpfunction,samplerate;
   int state,error,sec,hundsec,overload;
```

```
int monitor = 1;                /*  record monitor always on */

long int sz;
int  srate,sencode,sresol;
float sdur;

ERROR  err;
struct SND_HDR  hdr;


if ((read_snd_hdr(filename,&hdr)) == 0) /* read the  sound file header */
{

    vpfunction = BEGIN;             /* alert to the driver */

    port = 0x280;
    useint = 2;
    VP620 (&vpfunction, &useint, &port);


    if (hdr.s_samplrate == 8000)
         srate = 0;
    else
         srate = 1;

    vpfunction = PLAY;
    VP620 (&vpfunction, &srate, filename);  /*  ope.` file */

    vpfunction = STATUS;

do {
   VP620 (&vpfunction, &overload, &hundsec, &sec, &error, &state);
   }
while (!kbhit() & state != 3);

/*  these statements always required to close the file */
    vpfunction = STOP;
    VP620 (&vpfunction);

    vpfunction = END;
    VP620 (&vpfunction);
}
```

```
else
   {
   displayerr(READ);
   }
}

/* **************************** M A I N ********************************** */

main(argc,argv)                    /* receives sound file to play */
int argc;
char *argv[];
{
     SSantex_play(argv[1]);

} /* end main */

/* *********************************************************************** */
```

```
/**************************** SND_STRU.C ******************************
   This structure represents the sound object whose features will be
   stored in the sound file as a header file prefix to each recorded file.
   The database information will consist only of the unique file
   identifier and the description data.
***********************************************************************/

struct SND_HDR

  {

  long int  s_size;          /* number of bytes */
  int  s_samplrate;          /* 8K or 16K per sec */
  int  s_encoding;           /* 0=none,1=ADPCM */
  float  s_duration;         /* time in sec and hundredths */
  int  s_resolution;         /* # bits per sample  */

  } hdr_info;
```

```
/*************************** SND_ERRS.C ******************************
  This module contains a list of possible I/O error responses. This list
  is truly extensible.
 *******************************************************************/

typedef enum
  {
   PARS, WOPEN, WRITE, WCLOSE, ROPEN, READ, RCLOSE,
     SRATE, OK
  } ERROR;

void displayerr(e)

ERROR e;

{
  switch (e)

  {

   case PARS     : printf("Incorrect parameters\n");
                 return;
   case WOPEN    : printf("Cannot open file for output\n");
                 return;
   case WRITE    : printf("File write error\n");
                 return;
   case WCLOSE   : printf("Cannot close output file\n");
                 return;
   case ROPEN    : printf("Cannot open file for input\n");
                 return;
   case READ     : printf("File read error\n");
                 return;
   case RCLOSE   : printf("Cannot close input file\n");
                 return;
   case SRATE    : printf("Incompatible sampling rates for files\n");
                 return;

     }
}
```

```
/*************************** SND_NAME.C ********************************
  This program creates the first four digits of a file name for a recorded
  sound, by using the standard time parameters of GMT.
***************************************************************************/

/*------------------------ First Digit ------------------------------------*/
char YR(yr)
int yr;
{
  switch(yr)
  {
     case 90:  return '0';  break;      /*1990*/
     case 91:  return '1';  break;
     case 92:  return '2';  break;
     case 93:  return '3';  break;
     case 94:  return '4';  break;
     case 95:  return '5';  break;
     case 96:  return '6';  break;
     case 97:  return '7';  break;
     case 98:  return '8';  break;
     case 99:  return '9';  break;
     case 01:  return 'a';  break;
     case 02:  return 'b';  break;
     case 03:  return 'c';  break;
     case 04:  return 'd';  break;
     case 05:  return 'e';  break;      /*2005*/
  }
}

/* ------------------------- Second Digit ---------------------------- */
char MN(mn)
int  mn;
{
  switch (mn)
  {
     case 1:   return '1';  break;   /*January*/
     case 2:   return '2';  break;
     case 3:   return '3';  break;
     case 4:   return '4';  break;
     case 5:   return '5';  break;
     case 6:   return '6';  break;
     case 7:   return '7';  break;
     case 8:   return '8';  break;
```

```
        case 9:   return '9';  break;
        case 10:  return '0';  break;
        case 11:  return 'A';  break;
        case 12:  return 'B';  break;    /*December*/
    }
}

/* -------------------------- Third Digit ----------------------------- */

char DAY(day)
int  day;
{
    switch (day)
    {
        case 1:   return '1';  break;
        case 2:   return '2';  break;
        case 3:   return '3';  break;
        case 4:   return '4';  break;
        case 5:   return '5';  break;
        case 6:   return '6';  break;
        case 7:   return '7';  break;
        case 8:   return '8';  break;
        case 9:   return '9';  break;
        case 10:  return 'a';  break;
        case 11:  return 'b';  break;
        case 12:  return 'c';  break;
        case 13:  return 'd';  break;
        case 14:  return 'e';  break;
        case 15:  return 'f';  break;
        case 16:  return 'g';  break;
        case 17:  return 'h';  break;
        case 18:  return 'i';  break;
        case 19:  return 'j';  break;
        case 20:  return 'k';  break;
        case 21:  return 'l';  break;
        case 22:  return 'm';  break;
        case 23:  return 'n';  break;
        case 24:  return 'o';  break;
        case 25:  return 'p';  break;
        case 26:  return 'q';  break;
        case 27:  return 'r';  break;
        case 28:  return 's';  break;
        case 29:  return 't';  break;
```

```
        case 30:  return 'u';  break;
        case 31:  return 'v';  break;

    }
}


/* ---------------------------- Fourth Digit ------------------------------ */

char HR(hr)
int  hr;
{
   switch (hr)
   {
        case 1:   return '1';  break;
        case 2:   return '2';  break;
        case 3:   return '3';  break;
        case 4:   return '4';  break;
        case 5:   return '5';  break;
        case 6:   return '6';  break;
        case 7:   return '7';  break;
        case 8:   return '8';  break;
        case 9:   return '9';  break;
        case 10:  return 'a';  break;
        case 11:  return 'b';  break;
        case 12:  return 'c';  break;
        case 13:  return 'd';  break;
        case 14:  return 'e';  break;
        case 15:  return 'f';  break;
        case 16:  return 'g';  break;
        case 17:  return 'h';  break;
        case 18:  return 'i';  break;
        case 19:  return 'j';  break;
        case 20:  return 'k';  break;
        case 21:  return 'l';  break;
        case 22:  return 'm';  break;
        case 23:  return 'n';  break;
        case 24:  return 'o';  break;
   }
}
```

```
/************************ RECEIVER.C ********************************
  This program code is a modified version of a receiver socket application
  program in the Excelan "The LAN WorkPlace" Network Software for PC DOS
  Socket Library Application Program Interface manual. This program
  receives the sound file name sent by the sending socket in the MDBMS and
  stores it into a file.
*********************************************************************/
#include <sys/extypes.h>

#include <stdio.h>
#include <ctype.h>
#include <sys/exerrno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <signal.h>
#ifdef LATTICE
#include <error.h>
#else
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#endif

struct sockaddr_in recv_socket = { AF_INET };
struct sockaddr_in send_socket = { AF_INET };

#ifdef LATTICE
#define O_BINARY O_RAW
#endif
#define FILEOFLAG (O_CREAT | O_EXCL | O_WRONLY | O_BINARY)
#define FILEPMODE (S_IREAD | S_IWRITE)

#define MAXBUF        8192

extern int errno;
extern int break_enabled;
extern int abort_op;

int   fdr = -1;       /* receiver socket descriptor */
int   fdw = 1;        /* stdout for displaying message sent */
int   filefd = 0;          /* input data file, intialize to stdin */
char  buf[MAXBUF] = { 0 };/* buffer */
```

```
char  *filename = "";       /* input file name */
int   bufsize = 1024;       /* default buffer transfer size */
int   cnt = 0, netcnt = 0;  /* number of bytes read, sent */
short port = 2000;          /* port address for the connection */

int   break_handler();


main(argc, argv)
char **argv;
{
	int an;

	/* Check that the driver is loaded, and enable ^C handling */

	if (!loaded()) errexit("driver NOT loaded");
	signal(SIGINT, break_handler);
	break_enabled = 1;

	/* process arguments */

	for (an = 1; an < argc; ++an)
		if (argv[an][0] == '-') {
			switch (argv[an][1]) {
			case 'z':  /* set buffer si'z'e. */
				if (++an < argc) {
					bufsize = atoi(argv[an]);
					if ((bufsize <= 0) || (bufsize > MAXBUF))
					errexit("illegal buffer size");
					fprintf(stderr, "bufsize = %d\n", bufsize);
				}
				else errexit("expected buffer size");
				break;
			case 'f':  /* name of file to transfer */
				if (++an < argc) {
					filename = argv[an];
					filefd = open(filename, FILEOFLAG, FILEPMODE);
					if (filefd < 0)
					errexit("cannot create filename");
				}
				else errexit("expected filename");
				break;
			default:   /* unknown argument */
```

```
                errexit("unknown argument");
                break;
            } /* end of switch */
        } /* end of if */

    /* Set up the socket to receive data */

    recv_socket.sin_port = htons(port);
    fdw = filefd;

    /* Make a socket call */

    if ((fdr = socket(SOCK_STREAM, (struct sockproto *)0,
       &recv_socket, (SO_ACCEPTCONN | SO_KEEPALIVE))) < 0)
        errexit("socket");
    fprintf(stderr, "socket fd = %d\n", fdr);

    /* Accept a connection */

    fprintf(stderr, "posting accept fd = %d\n", fdr);
    if (accept(fdr, &send_socket) < 0)
        errexit("accept");
    fprintf(stderr, "accepted connect from partner\n");

    /* Read in the data from the socket and display it on the screen
       or write it to the file. */


        if ((netcnt = soread(fdr, buf, bufsize)) < 0) errexit("soread");
        fprintf(stderr, "read %d bytes\n", netcnt);

        if ((cnt = write(fdw, buf, netcnt)) < 0) errexit("write");

    if (fdw > fileno(stderr))
        close(fdw);
    soclose(fdr);
}

errexit(errstring)
char *errstring;
{
    if (errno) experror(errstring);
    else fprintf(stderr, "%s\nusage: receiver [-file name] [-zbuffer size]\n", errstring);
```

```
	if (fdr >= 0) soclose(fdr);
	exit(1);
}

break_handler()		/* break handler ... control-break or control-c */
{
	static int break_count = 0;

	if (++break_count == 1) {
		/* first time, just try to stop current network operation */
		abort_op = 1;
		signal(SIGINT, break_handler);/* reset trap */
		return;
	}
	else {
		/* second time, try to clean up, then quit */
		errexit("user abort");
	}
}
/*************************************************************************/
```

APPENDIX B - SUN SOUND MANAGEMENT PROGRAM CODE

The following programs are written for the setup to play a sound file in a remote PC and to read the text file transferred from PC to the SUN environment that contains the header information of the recorded sound. The main program is integrated into the main MDBMS program as modules, and one can find the main program code in [Pe90,Po90].

The program code is created or modified from the code written by Sawyer [Sa88] and the sending socket routine in the main program is a modified version of the "Internet Domain Stream Connection" program in the SUN Microsystem Network Programming manual, chapter 8 (A Socket-Based Interprocess Communications Tutorial).

```
/************************* SDEMO.C ***********************************
* Title       : SUN SOUND INTERFACE
* Author      : Yavuz Vural ATILA
* Rank        : LTJG Turkish Navy
* Advisor     : Prof. Vincent Y. LUM
* Date        : 5 September 1990
* Revised     : 20 October 1990
* Description : This program is designed to play the previously recorded
*               sound file in a remote PC, and to read the sound header
*               information from a text file which is transferred from PC
*               by using the file transfer protocol (FTP).
*********************************************************************/

#include <stdio.h>
#include <sys/types.h>
#include <time.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <netdb.h>
#include "snd_str.c"


#define SOUND_ERROR -1

/******************** SOCKET CONNECTION TO PC ***********************
 This part connects SUN to PC and sends the sound filename to play in PC.
*********************************************************************/

SendSocket(filename,pcname)

    char    *pcname;          /* remote PC host name */
    char    *filename;
{
    short    port = 2000;          /* virtual port number between SUN & PC */

    int sock;
    struct sockaddr_in server;
    struct hostent *hp, *gethostbyname();
    char buf[1024];

    /* Create socket */
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("opening stream socket");
        exit(1);
    }

    /* Connect socket using name specified by command line. */
    server.sin_family = AF_INET;
    hp = gethostbyname(pcname);
     if (hp == 0) {
        fprintf(stderr, "%s: unknown host\n", pcname);
        exit(2);
    }

    bcopy((char *)hp->h_addr, (char *)&server.sin_addr, hp->h_length);
    server.sin_port = htons(port);

    if (connect(sock,
        (struct sockaddr *)&server, sizeof server ) < 0) {
        perror("connecting stream socket");
```

```
            exit(1);
        }

        if (write(sock,filename,12) < 0)    /*gets the filename for playing*/
            perror("Writing on stream socket");
        close(sock);
        printf("\n\t\tPlease, press ENTER to continue...");
        return;
}

/********************** SOUND INFO READ*****************************
 This part reads the sound file header information from the given sound
 text file, which is already sent from PC to SUN.
 *******************************************************************/

 Snd_inforead(filename,r)

        char *filename;                     /* given input text file */
        struct SND_HDR *r;
  {
      FILE  *f;

       if ((f = fopen(filename,"r")) == NULL)        /* open for reading */
         {
           perror("Cannot open file for input");
           return SOUND_ERROR;
         }

       /*  read the header from the given input text file   */
            fscanf(f,"%s",r->sfname);
            fscanf(f,"%d",&r->s_size);
            fscanf(f,"%d",&r->s_samplrate);
            fscanf(f,"%d",&r->s_encoding);
            fscanf(f,"%f",&r->s_duration);
            fscanf(f,"%d",&r->s_resolution);
        fclose(f);
   return;
}

/*************************** MAIN ***********************************/

main()
```

```
{
 char filename[13];
 char answer, a;
 char pcname[7];
 struct SND_HDR *hdr;
 hdr= (struct SND_HDR *)malloc(sizeof(struct SND_HDR));

 /*gets the remote PC name like 'pclum1' or 'pclum2', to play sound*/
 printf("\n\tPlease ENTER << the remote PC name >> to play sound file\n");
 gets(pcname);

 while (answer != '0')  {

  answer = 0;

  while (answer < '0' || answer > '2') {

  printf("\n\n\t\t\tTHE SUN REMOTE SOUND PLAY MANAGEMENT\n");
   printf("\t*********************************************************");
   printf("\n\t\t1. Play Sound File in PC");
   printf("\n\t\t2. Display Sound File Header Information");
   printf("\n\t\t0. Exit\n");
   printf("\t*********************************************************");
   printf("\n\tEnter your choice number : ");
   answer = getchar();

   while (getchar() != '\n');
    printf("\nYour answer is %c\n",answer);

   switch (answer)

    {

    case '1' :

        printf("\n\t\t==== PLAY SOUND IN PC ====\n");
        puts("Enter the sound filename (xxxxxxxx.snd) to play in PC\n");
        gets(filename);
        SendSocket(filename,pcname);    /*play sound file in remote PC*/
        a= getchar();
        break;

    case '2' :
```

```
            printf("\n\t\t====DISPLAY SOUND FILE HEADER INFO====\n\n");
            puts("Enter the 'USER DEFINED' sound text file name\n");
            gets(filename);
            snd_inforead(filename,hdr);  /*read the header info*/


            /* returned header info from snd_info read function */
            printf("\n\tFILENAME   : %s\n",hdr->sfname);
            printf("\tSIZE       : %d\n",hdr->s_size);
            printf("\tSAMPLERATE: %d\n",hdr->s_samplrate);
            printf("\tENCODING   : %d\n",hdr->s_encoding);
            printf("\tDURATION   : %f\n",hdr->s_duration);
            printf("\tRESOLUTION: %d\n",hdr->s_resolution);

            printf("\nPlease, press ENTER to continue...");
            a= getchar();
            break;

         case '0' :

            printf("\n\n\n\t\t\t**************************************\n");
            printf("\t\t\t============ THANK YOU ============\n");
            printf("\n\t\t\t========= HAVE A NICE DAY =========\n");
            printf("\t\t\t**************************************\n");
            break;

       } /* end switch */
     } /* end while ans not in 0..1 */
   } /* end while ans not '0' */
} /*end main*/


/**************************************************************************/
```

```
/***************************** SND_STR.C ******************************
 This structure represents the sound object header information, which
 is transferred from PC to SUN environment to store the sound object
 registration information to the MDBMS tables.
*********************************************************************/

struct SND_HDR

  {

  char sfname[13];           /* unique sound filename */
  int  s_size;               /* number of bytes */
  int  s_samplrate;          /* 8K or 16K per sec */
  int  s_encoding;           /* 0=none,1=ADPCM */
  float  s_duration;         /* time in sec and hundredths */
  int  s_resolution;         /* # bits per sampie  */

  } hdr_info;
```

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Center for Naval Analysis 4401 Ford Ave. Alexandria, Virginia 22302-0268	1
4.	John Maynard Code 042 Command and Control Departments Naval Ocean Systems Center San Diego, California 92152	1
5.	Dr. Sherman Gee ONT-221 Chief of Naval Research 880 N. Quincy Street Arlington, Virginia 22217-5000	1
6.	Leah Wong Code 443 Command and Control Departments Naval Ocean Systems Center San Diego, California 92152	1
7.	Professor Vincent Y. Lum Code CsLm Department of Computer Science Naval Postgraduate School Monterey, California 93943	2

8.	Deniz Kuvvetleri Komutanligi Personel Daire Başkanligi Bakanliklar, Ankara / TURKEY	1
9.	Deniz Harp Okulu Komutanligi 81704 Tuzla, Istanbul / TURKEY	2
10.	Yavuz Vural ATILA Dz.Kd.Utgm Selami Ali Mahallesi Yeni Dershane Sokak, 8/3 81140 Uskudar, Istanbul / TURKEY	2
11.	Professor David Hsiao Code CsHq Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
12.	Professor C. Thomas Wu Code CsWq Department of Computer Science Naval Postgraduate School Monterey, California 93943	2
13.	Professor Klaus Meyer-Wegener University of Erlangen-Nuernberg IMMD VI, Martensstr.3, 8250 Erlangen / GERMANY	1
14.	Dr. Bernhard Holtkamp University of Dortmund Department of Computer Science Software Technology P.O. Box 500 500 D-4600 Dortmund 50 / GERMANY	1